Jan Gazda
Posted on Sep 1, 2020

AWS Glue first experience - part 5 - Glue Workflow, monitoring and rants

#aws #datascience #python #serverless

My AWS Glue journey (5 Part Series)

1 AWS Glue first experience - part 1 - How to run your code?

2 AWS Glue first experience - part 2 - Dependencies and guts

3 AWS Glue first experience - part 3 - Arguments & Logging

4 AWS Glue first experience - part 4 - Deployment & packaging

5 **AWS Glue first experience - part 5 - Glue Workflow, monitoring a...**

In this episode, we are going to look at AWS Glue Workflow, mention time-consuming tasks during development and wrap up.

Challenge number 7: the Workflow

To define ETL pipelines AWS Glue offers a feature called Workflow, where you can orchestrate your Crawlers and Jobs into a flow using predefined triggers. Workflow provides a visual representation of your ETL pipeline and some level of monitoring.

This, in theory, is a nice thing to have for your ETL pipeline however I discovered a lot of problems prevented me from using it's promised potential.

Definition

Defining the workflow via the AWS Console is quite simple.

The user interface resembles AWS CloudFormation template Designer however with a very limited set of features.

Adding your jobs and triggers to the workflow graph feel quite bulky because the graph screen does not scale well in the browser (at least on 13" MacBook).

Due to the number of required parameters, each click brings up a modal window where you need to fill something or select. The navigation supports mouse only and there aren't any keyboard shortcuts supported which require a lot clicking around, especially if you just want to remove a few nodes. It is not possible to select multiple components or delete the previous chain.

Delete action on triggers is immediately performed on an actual resource which can take quite some time. Workflow graph is automatically saved after each activity which can be potentially dangerous if one is not careful when editing if the workflow is triggered by the cron.

Also undo and redo buttons are not present so the only way to recover modified workflow is from the previous run (if there was any).

Defining the workflow with IaC either Terraform or AWS CloudFormation gets much more difficult. Just due to the fact that your workflow is defined as a set of trigger resources bound to the workflow resource, where each trigger contains a reference to the Job or Crawler resource, so the final flow not really obvious. So the best way to define the workflow is to use the user interface and then export it using `aws glue get-workflow --name` or same API function. Then adapt the graph JSON into your IaC code.

State management

Like jobs have `DefaultArguments`, the workflow has run properties which can be accessed or modified prior to or during the workflow run. This provides a good way of sharing the state or specific config.

Workflow properties are submitted to your job via system arguments `'--WORKFLOW_NAME', 'my-workflow', '--WORKFLOW_RUN_ID', 'wr_1e927676cc826831704712dc067b46480042ca7aead5caa6bea4fc587311e29d'`. Yet again this was inconsistent between PySpark and Python Shell jobs but seems it's not anymore.

Execution & Monitoring

There are several ways to start the workflow but the first node has to be the trigger `on-demand` or `schedule` (cron). In case you select `event` and add your job as a first node, workflow won't start.

AWS Glue Workflow Console view

When the workflow is started (manually or by the trigger) it still takes some time to kick off the first job (I experienced times between 1 - 4 minutes). There is also a delay in between jobs during the workflow run which is around 1 minute.

From my observation, it looks like that workflow appears to be a pull-based system instead of the expected event-based system.

Which also means that any events you'd like to know about eg. workflow started, the job started, job finished, have to be implemented inside the code of the job itself.

This also brings me to another point that workflow is not able to provide information whether it finished successfully or not. The only states you can get at the moment are

`'RUNNING'|'COMPLETED'|'STOPPING'|'STOPPED'`. To find out which job or crawler failed via AWS Console you have to navigate to the `Workflows`, select workflow and then `History` tab, then select `workflow run` you want to investigate and click `View run details`. After that, you can see which job has failed.

In case your workflow has been running for some time and accumulated a lot of `workflow runs` rendering of the `History` tab can take a noticeable amount of time.

To investigate the workflow via CLI/API with a function `get_workflow_run` (include graph), you then have to iterate through two arrays `JobDetails.JobRuns` and `CrawlerDetails.Crawls` to find out what failed.

Challenge number 8: Time

As a developer, your most valuable thing is your time and ability to iterate as fast and as much as possible. With AWS Glue you should prepare yourself for quite long breaks.

This is a rough summary of how long take certain operations.

- Start PySpark Glue 1.0 job - it takes a minimum 10 minutes to start the cluster and run your job. And then a minute or two to run your code. This can be really frustrating if you've made a typo which wasn't caught by the linter.
- Start PySpark Glue 2.0 job - Fortunately the startup time has been significantly reduced and it took roughly 4-6 minutes (often much less) to run the code.
- Start Python Shell Glue 1.0 job - this takes roughly 2 minutes to start the execution of your code
- Preparing dependencies - If you decide not to use online editor, you will be required to perform a lot of operations to get your code up to the cloud. So this was the first task to automate.
- Checking logs in AWS CloudWatch Logs - there is another ~1-2 minutes delay before the logs are populated
- Workflow run takes longer than expected due to polling mechanism.

Summary

This was my first experience with AWS Glue ETL service.
In general, it was a good experience and it is a fully managed service it simplifies a lot, especially when it comes to creating and maintenance of a cluster.

The service is being actively developed and things may change without you knowing. It is nice in general but has yet a long way to go to become more user and developer-

friendly.

Here is the list of my grumbles:

- The inability to update C based libraries (NumPy/Pandas) in PySpark jobs may pose a problem.
- Packaging and defining and supplying dependencies for the jobs is clumsy.
- Startup time of PySpark Glue 1.0 is really long.
- The inconsistency in API and parameters prevents better implementations.
- Unnecessary workaround needed to implement optional arguments.
- Open-source code may not be up to date.
- Workflow interface and definition could have been more intuitive.
- Event-driven or extended functionality of the workflow must be developed outside Glue service eg. using AWS Lambda and step functions.

From the future versions of AWS Glue, I'd like to see little more focus on development experience.
Especially addressing the inconsistent API between jobs, packaging/deployment and AWS Glue lib maintenance.

The code for the examples in this article can be found in my GitHub repository aws-glue-monorepo-style

My AWS Glue journey (5 Part Series)

1	AWS Glue first experience - part 1 - How to run your code?
2	AWS Glue first experience - part 2 - Dependencies and guts
3	AWS Glue first experience - part 3 - Arguments & Logging
4	AWS Glue first experience - part 4 - Deployment & packaging
5	**AWS Glue first experience - part 5 - Glue Workflow, monitoring a...**

Top comments (2)

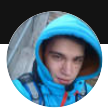
daniel ortega • Sep 1 '20

Hi Jan
Nice post, thanks for sharing. I kinda agree with your summary here so I will recommend people working with Glue to read this series of posts.
Well done.

Also I am taking the opportunity to say hello, long time no see you!
BR
D

Jan Gazda • Sep 2 '20 • Edited on

Hi Daniel!
It's been a long time indeed.
I'm glad this is useful for others even though there is a plenty room for an improvement.



Jan Gazda

I solve problems, usually with Python. Organize PyAmsterdam python meetups.

LOCATION
Amsterdam

WORK
Freelance engineer

JOINED
Jul 10, 2020

More from Jan Gazda

AWS Glue first experience - part 4 - Deployment & packaging

#aws #python #datascience #serverless

AWS Glue first experience - part 3 - Arguments & Logging

#aws #python #datascience #serverless

AWS Glue first experience - part 2 - Dependencies and guts

#aws #datascience #python #serverless