Jan Gazda
Posted on Aug 28, 2020 • Updated on Aug 31, 2020

AWS Glue first experience - part 3 - Arguments & Logging

#aws #python #datascience #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 an...

Challenge number 3: Arguments & Config

Almost every application requires some kind of config or parameters to start with the expected state, AWS Glue applications are no different.

Our code is supposed to run in 3 different environments (accounts), DEV, TEST, PROD and several configuration values were required eg. log level, SNS topic (for status updates) and few more.

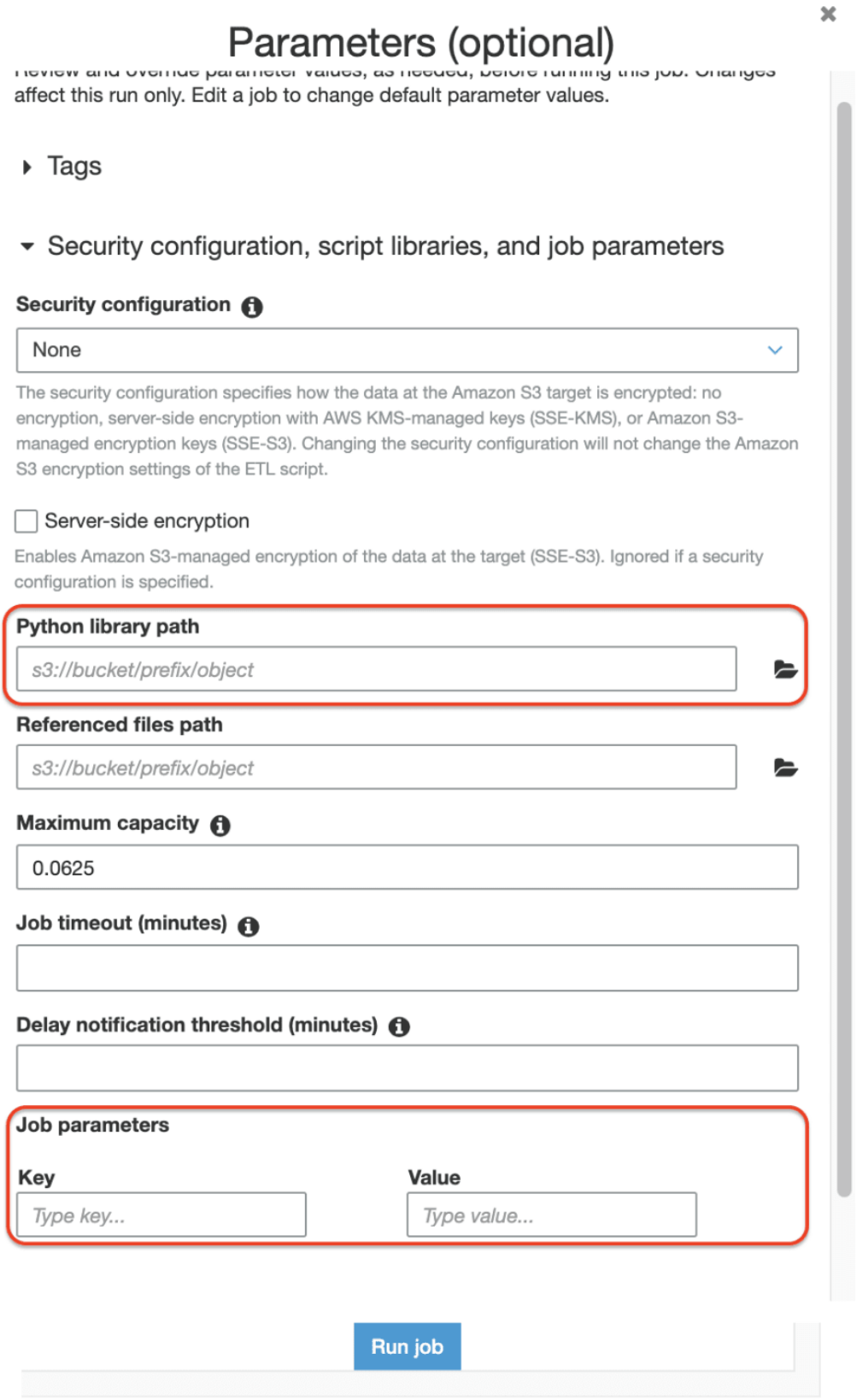
Documentation mentions special parameters however they are not all arguments you can expect to get. We will explore this later in this section.

During my work on a project, there was only one set of `DefaultArguments` which could have been overwritten prior to job start. By the time of writing this article, there are now two sets of these `DefaultArguments` and `NonOverridableArguments` where the latter has been added recently.

Some of these arguments were supplied as SSM Parameters while others were submitted as `DefaultArguments`. This could be very useful in case the job fails and we'd like to run the job again with a different log level eg. default `WARN` vs non-default `DEBUG`.

To add or change an argument of the job prior to it's run you can either use a console

```
Security configuration, script libraries, and job parameters -> Job parameters
```



AWS Glue Job parameters

Or when using CLI/API add your argument into the section of `DefaultArguments`.

Then inside the code of your job you can use built-in `argparse` module or function provided by [aws-glue-lib](#) [getResolvedOptions](#) (`awsglue.utils.getResolvedOptions`).

When I started with my journey the function `getResolvedOptions` was not available for Python Shell jobs and I also planned to create a config object which holds the necessary configuration for the job. It got implemented later.

There is a difference between the implementation of `getResolvedOptions` between `awsglue` present in PySpark jobs and `awsglue` present in Python Shell jobs.

The code of `awsglue` used in PySpark jobs can be located at GitHub inside aws-glue-lib repository. The main difference is that PySpark job handles some cases of reserved arguments

The code used inside Python Shell jobs is this.

```
# This code has been extracted from
# AWS Glue 1.0 Python Shell runtime awsglue.utils.getResolvedOptions

import argparse


def getResolvedOptions(args, options):
    parser = argparse.ArgumentParser()
    for option in options:
        parser.add_argument('--' + option, required=True)
    parsed, extra = parser.parse_known_args(args)
    return vars(parsed)
```

pyshell_getResolvedOptions.py hosted with ❤ by GitHub view raw

The main problem of this function is that it makes all `DefaultArguments` **required**. Which is rather clumsy considering the fact that it also requires you to use `--` (double dash) in front of your argument which is generally used for optional arguments.

It is possible to re-implement optional argument this by wrapping this function as suggested in this StackOverflow answer. However, this is rather a workaround which may break if the AWS team decides to fix this.

Also when specifying `DefaultArguments` via console it feels more natural not to include `--` as the UI is not mentioning this at all.

Missing arguments in sys.argv

My first few jobs were only using PySpark and I discovered that there are some additional arguments present in `sys.argv` which are used in examples inside developers guide but not described. To get a description of these arguments one

should visit AWS Glue API docs page which is a bit hidden because there is only 1 direct link pointing there from the developers' guide.

Here are arguments present in `sys.argv` for PySpark job (Glue 1.0).

```
[
  'script_2020-06-24-07-06-36.py',
  '--JOB_NAME', 'my-pyspark-job',
  '--JOB_ID', 'j_dfbe1590b8a1429eb16a4a7883c0a99f1a47470d8d32531619babc5e283df
  '--JOB_RUN_ID', 'jr_59e400f5f1e77c8d600de86c2c86cefab9e66d8d64d3ae937169d766
  '--job-bookmark-option', 'job-bookmark-disable',
  '--TempDir', 's3://aws-glue-temporary-<accountID>-us-east-1/admin'
]
```

Parameters `JOB_NAME`, `JOB_ID`, `JOB_RUN_ID` can be used for self-reference from inside the job without hard coding the `JOB_NAME` in your code.

This could be a very useful feature for self-configuration or some sort of state management. For example, you could use `boto3` client to access the job's connections and use it inside your code. Without specifying the connection name in your code directly. Or if your job has been triggered from the workflow it would be possible to refer to the current workflow and its properties.

Let's explore `sys.argv` of Python Shell jobs

```
[
  '/tmp/glue-python-scripts-7pbpva1h/my_pyshell_job.py',
  '--job-bookmark-option', 'job-bookmark-disable',
  '--scriptLocation', 's3://aws-glue-scripts-133919474178-us-east-1/my_pyshell
  '--job-language', 'python'
]
```

Above we can see that set arguments available in Python Shell job.

The arguments are a bit different from what we've got in PySpark job but the major problem is that arguments `JOB_NAME`, `JOB_ID`, `JOB_RUN_ID` are not available.

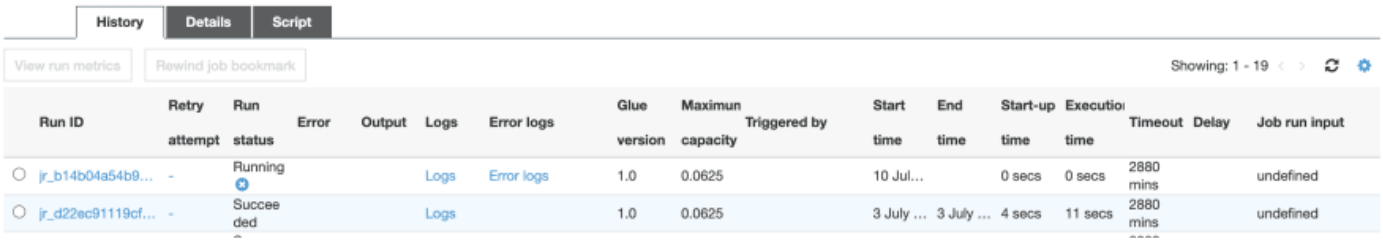
This creates a very inconsistent developer experience and **prevents** the self-reference from inside the job which diminishes the potential of these parameters.

Challenge number 4: Logging

Like I already mentioned AWS Glue Job logs are sent to AWS CloudWatch logs.

There are two log groups for each job. `/aws-glue/python-jobs/output` which contains the `stdout` and `/aws-glue/python-jobs/error` for `stderr`. Inside log groups you can find the log stream of your job named with `JOB_RUN_ID` eg. `/aws-glue/python-jobs/output/jr_3c9c24f19d1d2d5f9114061b13d4e5c97881577c26bfc45b99089f2e1abe13cc`.

When the job is started there are already 2 links helping you to navigate to the particular log.



Showcase of aws console.

Even though the links are present, the log streams are not created until the job starts.

When using logging in your jobs, you may want to avoid logging to `stderr` or redirect it to `stdout` because `error` log stream is **only** created when the job finishes with failure.

Glue 1.0 PySpark job logs are very verbose and contain a lot of "clutter", unrelated to your code. This clutter comes from Spark underlying services. This issue has been addressed in Glue 2.0 where the exposure to the logs of unrelated services is minimal and you can comfortably focus on your own logs. Good job AWS Team!

Python Shell jobs do not suffer from this condition and you can expect to get exactly what you log.

And that's it about the config and logging. In the next episode, we are going to look into packaging and deployment.

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 an...

Top comments (1)

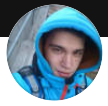
ramasamykannan • Jan 15 '21

Hi,

I have a doubt here.

We want to maintain a config file in aws glue where we want to maintain some parameters which are common for all the jobs, and we want refer those parameter values with in the glue jobs. Is that possible ?

Thanks,
Ramasamy kannan.



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 5 - Glue Workflow, monitoring and rants

#aws #datascience #python #serverless

AWS Glue first experience - part 4 - Deployment & packaging

#aws #python #datascience #serverless

AWS Glue first experience - part 2 - Dependencies and guts

#aws #datascience #python #serverless