Jan Gazda
Posted on Aug 25, 2020 • Updated on Aug 31, 2020

AWS Glue first experience - part 2 - Dependencies and guts

#aws #datascience #python #serverless

My AWS Glue journey (5 Part Series)

Challenge number 2: Dependencies

In the previous episode, we have learned that it's rather simple to put the code in one python file and run it. However, this is often not the best solution and it makes more sense to split the code into separate modules or use the code from the existing libraries. I was about to call this part external dependencies but you will soon find out why I left the `external` out.

There are multiple documentation pages mentioning how to work with dependencies and differences between `PySpark` and `Python Shell` jobs. This may not be really obvious when you browse the docs for the first time.

For PySpark jobs, there is this a page:
https://docs.aws.amazon.com/glue/latest/dg/aws-glue-programming-python-libraries.html
For Python Shell jobs there is another page:

https://docs.aws.amazon.com/glue/latest/dg/add-job-python.html#python-shell-supported-library

Inconsistencies: Each of the pages can be found in completely different sections and therefore easily missed.

Providing dependencies for PySpark Jobs

English isn't my native language and documentation has confused me here with an explanation instead of giving a simple example.

At the moment Glue supports only pure python libraries which means we are not able to use C based libraries (`pandas` , `numpy`) or extensions from other languages.

The dependencies can be supplied in two forms.

a. Single `.py` file
b. `.zip` an archive containing python packages.

Packages inside the `.zip` archive needs to be in the root of the archive.

```
$ zipinfo -1 dependencies.zip
pkg2/
pkg2/__init__.py
pkg1/
pkg1/__init__.py
```

Each dependency has to be uploaded to S3 Bucket and then supplied as an " `Special parameter` _" `--extra-py-files` to the particular job in a format of comma-separated S3 URLs.

You have to specify each file separately `s3://bucket/prefix/lib_A.zip,s3://bucket_B/prefix/lib_X.zip` which becomes annoyingly clumsy if your job has more than one dependency.

Behind the scenes

Your `.py` or `.zip` file is copied into a `/tmp` directory accessible during the job runtime. Injected into a `PYTHONPATH` and also added as an argument `--py-files` to `spark-submit` .

Note:

> We have been given access to the Glue 2.0 preview which has broken the `spark-submit` argument and Spark did could not see the code provided inside the zip file. Fortunately, it is possible to supply the `py-files` during the job runtime to the `SparkSession`. This has been later fixed by AWS team and we were notified about the fix when our jobs started to fail due to applied workaround above. It's worth mentioning that Glue 2.0 is running Python 3.7.

Providing dependencies for Python Shell Jobs

Beware:

> The documentation of Python Shell jobs is really tricky and sometimes confusing mainly because the examples are provided without enough context and some code examples are written in legacy python while others in Python 3.

To provide external dependencies documentation advise using `.egg` or `.whl` files. Upload them to S3 and list as `–extra-py-files` argument.

The runtime environment includes several pre-installed packages.

```
Boto3
collections
CSV
gzip
multiprocessing
NumPy
pandas (required to be installed via the python setuptools configuration, setu
pickle
PyGreSQL
re
SciPy
sklearn
sklearn.feature_extraction
sklearn.preprocessing
xml.etree.ElementTree
zipfile
```

Due to my past experience with AWS Lambda, I suspected that versions of pre-installed packages may be outdated.

So I decided to create a simple job `my-pyshell-job` and edit the code using the online editor, just to get versions of specific libraries. The code is very simple:

```python
import boto3
import numpy
import pandas


print('boto3', boto3.__version__)
print('numpy', numpy.__version__)
print('pandas', pandas.__version__)
```

my_pyshell_job.py hosted with ❤️ by GitHub view raw

The result of the code above

```
boto3 1.9.203
numpy 1.16.2
pandas 0.24.2
```

confirmed that my assumption was correct and the versions are a bit dated so you won't get the latest stable versions.

Which left me no other choice than installing it.

This is the point where the documentation is lacking an explanation and I was left with trial and error approach.

The installation

Knowing that I need to provide `.whl` I ran the command `pip wheel pandas` on my macOS. And started to experiments.

This has downloaded multiple wheel files:

```
numpy-1.19.0-cp36-cp36m-macosx_10_9_x86_64.whl
python_dateutil-2.8.1-py2.py3-none-any.whl
six-1.15.0-py2.py3-none-any.whl
pandas-1.0.5-cp36-cp36m-macosx_10_9_x86_64.whl
pytz-2020.1-py2.py3-none-any.whl
```

I took my sample code from above and called it `my_pyshell_job.py` and together with the `.whl` files above uploaded to S3 bucket.

Updated a job with AWS CLI command.

```
  aws glue update-job --job-name my-pyshell-job --job-update '{
    "Role": "MyGlueRole",
    "Command": {"Name": "my-pyshell-job", "ScriptLocation": "s3://bucket/my_py
    "DefaultArguments": {
      "--extra-py-files": "s3://bucket/dependencies/library/numpy-1.19.0-cp36-
    }
  }'
```

And started the job via console.

The job finished with error `ImportError:`.

AWS Glue uses 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`. I also found out that `error` log is only created if the job finishes with error so if you are planning to log errors which won't fail the job log it to `stdout`.

The content of the `output` log contains `pip` logs. And reported that all libraries were successfully installed. However, the `error` log contains the whole stack trace of this error.

```
ModuleNotFoundError: No module named 'numpy.core._multiarray_umath'`.

ImportError:

IMPORTANT: PLEASE READ THIS FOR ADVICE ON HOW TO SOLVE THIS ISSUE!

Importing the numpy C-extensions failed. This error can happen for
many reasons, often due to issues with your setup or how NumPy was
installed.
```

The error message tells us that this is most likely caused by OS mismatch.

So I ran another simple job to give me more details about the platform.

```python
import platform
print(platform.platform())
```

print_platform.py hosted with ❤ by GitHub view raw

I tried this with both PySpark and Python Shell jobs and the results were a bit surprising.

Python Shell jobs run on `debian`: `Linux-4.14.123-86.109.amzn1.x86_64-x86_64-with-debian-10.2`
while PySpark jobs run on Amazon Linux `Linux-4.14.133-88.112.amzn1.x86_64-x86_64-with-glibc2.3.4` likely to be a `amazoncorretto`.

The next step was clear, I needed a `wheel` with `numpy` built on `Debian` Linux. Since the plan was to deploy using CI/CD I decided to use official python docker image `python:3.6-slim-buster`.

Run `pip wheel pandas` again to get the correct packages and remove those already installed.

```
numpy-1.19.0-cp36-cp36m-manylinux2010_x86_64.whl
pandas-1.0.5-cp36-cp36m-manylinux1_x86_64.whl
```

Upload `wheels` to S3.
Update and run the job.

The job has PASSED!
And this is how the result looks:

```
boto3 1.9.203
numpy 1.19.0
pandas 1.0.5
```

But wait there is more in the log.

```
Processing ./glue-python-libs-psoetpzo/numpy-1.19.0-cp36-cp36m-manylinux2010_x
Installing collected packages: numpy
Successfully installed numpy-1.19.0
Processing ./glue-python-libs-psoetpzo/pandas-1.0.5-cp36-cp36m-manylinux1_x86_
Collecting pytz>=2017.2
Downloading pytz-2020.1-py2.py3-none-any.whl (510 kB)
Collecting numpy>=1.13.3
Downloading numpy-1.19.0-cp36-cp36m-manylinux2010_x86_64.whl (14.6 MB)
Collecting python-dateutil>=2.6.1
Downloading python_dateutil-2.8.1-py2.py3-none-any.whl (227 kB)
Collecting six>=1.5
Downloading six-1.15.0-py2.py3-none-any.whl (10 kB)
Installing collected packages: pytz, numpy, six, python-dateutil, pandas
Successfully installed numpy-1.19.0 pandas-1.0.5 python-dateutil-2.8.1 pytz-20
```

If you look closely, you can notice that `numpy` has been installed twice, the question is why?

The answer lies inside another file called `/tmp/runscript.py`. This file is responsible for orchestrating the installation and running your code as well as providing the stack trace in case the exception.

/tmp/runscript.py

Out of pure curiosity, I decided to print out the content of this file to find out why `numpy` has been installed twice.

The content of this file was enlightening but also shocking.

After the first 12 lines of imports, there was a comment on line 14

```
##TODO: add basic unittest
```

Okay, this is a bit scary because a comment like this either suggests that developer did not have enough time to write the tests when rushing into production and the reviewer left this to slip.

If your code does not contain enough tests you should keep it internally or loudly warn your customers about potential consequences.

By further examination, I noticed that this script is also used for PySpark jobs.

Little further into the file, there is a function `download_and_install`. The comment above function describes it's purpose `# Download extra py files and add them to the python path` (might as well have used the docstring right?).

The function takes each file supplied in `--extra-py-files` argument and installs it separately with `pip`.

This tells me a couple of things:

1. Dependencies mentioned in `setup.py` are automatically downloaded from PyPI, so since `pandas` depend on `numpy` it will download it as well hence the second installation.
2. Using a private package repository where dependencies are located there as well won't work flawlessly.
3. The order of dependencies supplied matters

AWS Glue examples and libs

Speaking of dependencies AWS Glue provides its core functionality via a library called awsglue. The code is located on GitHub. Even though the code is public, the repository maintainers do not seem to be interested in community ideas and pull requests because there are many pull requests without any kind of response from AWS Glue team. Judging from the reactions on the unmaintained issue I am not the only one who feels this way.

Friendly warning, if you are going attempt anything using this repository on your local machine I strongly suggest doing it in an isolated environment either using `docker` container or VM. The code modifies some local directories and `PYTHONPATH`.

During my work on this project, the repository contained 2 versions one in each branch `glue-0.9` and `glue-1.0` + readme file which was a bit confusing because it described the process for both versions.

The "installation" process is also somewhat painful if the only thing you want is code completion in your IDE because the library does not provide any mechanism for a simple installation. And the only way how to run this code is to zip `awsglue` directory `zip -r PyGlue.zip awsglue`. And add it to your `PYTHONPATH` (see the `bin/setup.sh` for example). As you may have noticed the repository contains `bash` commands and Windows is not currently supported (there is an open pull request).

In case you are visiting the repository to look for the examples of tests you will be very disappointed because there aren't any tests at all!

Coding practices

Since the repository does not contain any kind of automation for linting and testing there are a lot of lint errors and potentially dangerous lines.

One example is using immutable as a default argument:

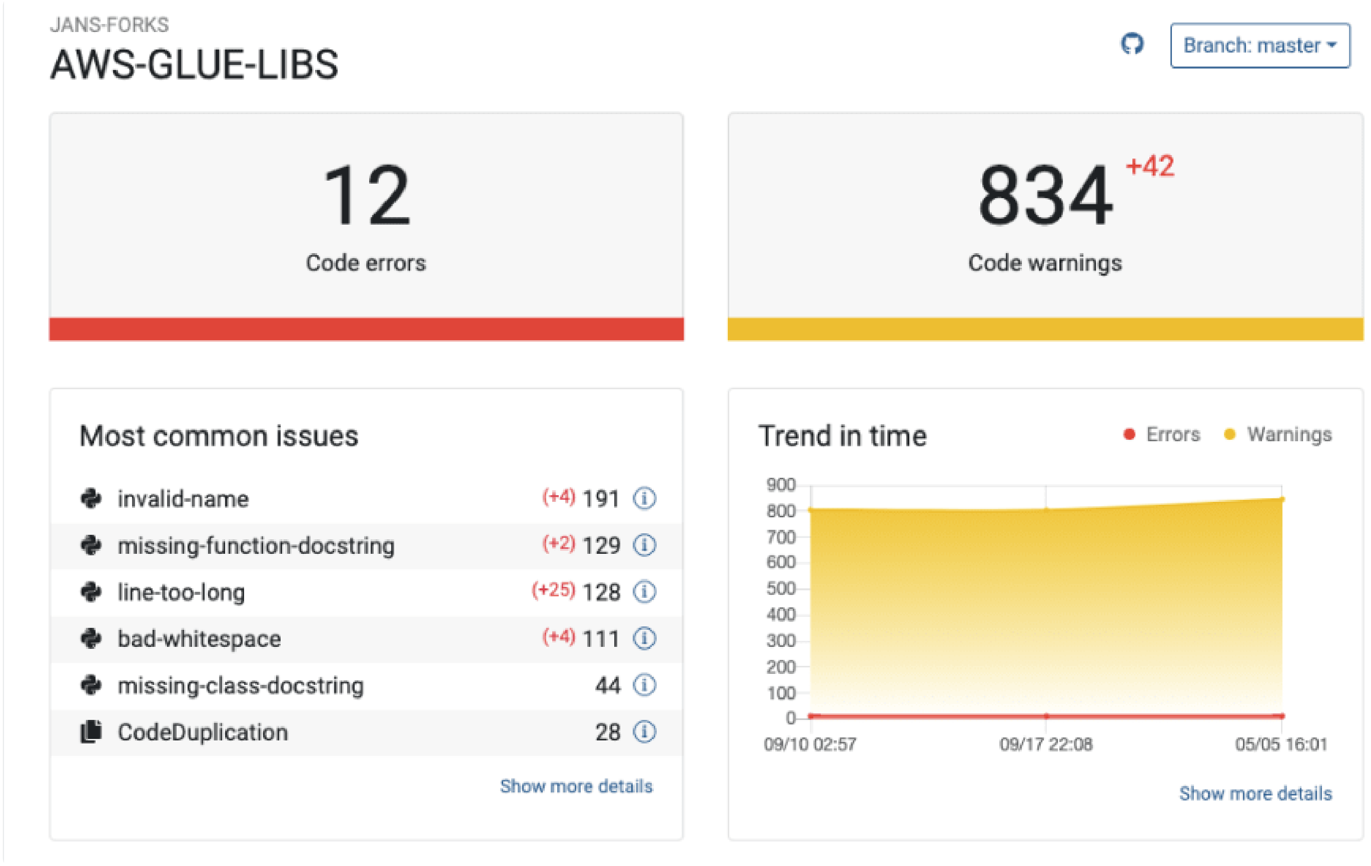
```
# taken from https://github.com/awslabs/aws-glue-libs/blob/0825fdafc76aadffa3d7846aaf867f25

class DataType(object):
    def __init__(self, properties={}):
        self.properties = properties
```

glue_immutable.py hosted with ❤ by GitHub view raw

This is the score of default pylint configuration `Your code has been rated at 2.94/10` Using a plain text and pylint score is not always ideal when presenting the errors to a wider audience. With codeac.io we can see the report in numbers but also track issue counts over time.

I forked the repository and run the report for you.
Please keep in mind that I left all configuration on default and therefore the results might not be 100% accurate but should give the idea about the current state.


JANS-FORKS
AWS-GLUE-LIBS

12 Code errors

834 +42 Code warnings

Most common issues

invalid-name	(+4)	191
missing-function-docstring	(+2)	129
line-too-long	(+25)	128
bad-whitespace	(+4)	111
missing-class-docstring		44
CodeDuplication		28

Results of codeac.io for aws-glue-libs

Apart from dangerous default values, you can find missing docstrings, `#TODO` comments, too long lines, unused imports and many others.

Before the latest update on May 5, 2020, there were several import errors preventing the code from even running or testing it. This update has patched a few holes however there is still a mix of imports present waiting to cause more problems. If you are curious about how python imports work I suggest reading this article from realpython

AWS maintains another repository called aws-glue-samples. The story of this repository is slightly better, it appears to be actively maintained I reported and issue about bad coding practices of using `import *` and received a response 2 days later what I consider a rather quick action in an open-source world.

Although I did not find examples particularly useful for myself and the code quality is a bit better than mentioned `aws-glue-libs`. The examples usually contain a lot of comments which could be really useful if you are
new to AWS Glue and data science in particular.

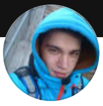
As you can see trivial example of executing one script file without dependencies can quickly escalate and catch you off guard. In the next episode, we are going to explore how to parametrise and configure the glue application.

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

Top comments (1)

Mauro Mascia • Feb 8 '21 • Edited on

Really interesting post, thank you very much.
I have just a point about the glue_immutable in part2: it should be mutable (as it is an object) not immutable. More on florimond.dev/blog/articles/2018/0...



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 3 - Arguments & Logging
#aws #python #datascience #serverless