



Jan Gazda

Posted on Aug 29, 2020 • Updated on Aug 31, 2020

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

#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...

In this episode, we are going to explore how can we reuse our code and how to deploy AWS Glue application which consists of more than one file.

I expected the workflow to be very similar to AWS Lambda which is already well known and optimised for python but due to involvement of Spark this is not entirely true for AWS Glue.

## Challenge number 5: Stay DRY

Because the initialisation process for all data sources and jobs was similar I decided it would be a good idea not having to repeat myself much and create a library with a set of functions which take care of parsing arguments, getting configuration data, simplify the PySpark or Pandas interface.

Due to the nature of different types of dependencies each job type requires. PySpark - `.py`, `.zip`, Python Shell - `.egg`, `.whl`. And the fact that all our code is held in monorepo.

I decided to create a simple python package with `setuptools` and follow the [src structure](#).

This gives me enough flexibility to produce needed formats and also reference to the library from inside `requirements.txt`.

```
1  from setuptools import setup, find_packages
2
3  setup(
4      name="glue_shared",
5      author="Jan Gazda",
6      author_email="jan.gazda@cloudreach.com",
7      python_requires=">=3.6",
8      classifiers=[
9          "Intended Audience :: Developers",
10         "Natural Language :: English",
11         "Programming Language :: Python :: 3.6",
12         "Programming Language :: Python :: 3.7",
13         "Programming Language :: Python :: 3.8",
14     ],
15     description="Helper library for AWS Glue jobs.",
16     setup_requires=["wheel"],
17     package_dir={"": "src"},
18     packages=find_packages(where="src", exclude=["contrib", "docs", "tests"]),
19     test_suite="tests",
20     version="0.0.1",
21     zip_safe=True,
22 )
```

simple\_setup.py hosted with ❤ by GitHub

[view raw](#)

## Challenge number 6: Deployment & packaging

Okay so now that I have all the necessary components covered let's put them together and deploy with [Terraform](#).

For each data source, we have defined two transitions `raw to refined` and `refined to curated`.

AWS Glue requires 1 `.py` file as an entry point and rest of the files must be plain `.py` or contained inside `.zip` or `.whl` and each job should be able to have a different set of requirements.

Another requirement from AWS Glue is that entry point script file and dependencies have to be uploaded to S3.

Anything uploaded to S3 then also has to be listed in [Terraform](#) as a [Special parameter](#) `--extra-py-files` in form of comma separated list of S3 URLs, eg.

```
s3://bucket/dep1.zip, s3://bucket/deb2.zip or s3://bucket/dep1.whl,  
s3://bucket/deb2.whl.
```

Since this list can be very dynamic it's best to keep it as short as possible. As you can see there are a number of operations require from the developer and developing more than one job requires a significant effort. Therefore I decided to use the following structure

```
/
├── glue/
│   ├── data_sources/
│   │   ├── ds1/
│   │   │   ├── raw_to_refined/
│   │   │   │   ├── Makefile
│   │   │   │   ├── config.py
│   │   │   │   ├── raw_to_refined.py
│   │   │   │   └── requirements.txt
│   │   │   └── refined_to_curated/
│   │   │       ├── Makefile
│   │   │       ├── config.py
│   │   │       ├── another_dependency.py
│   │   │       ├── refined_to_curated.py
│   │   │       └── requirements.txt
│   └── shared/
│       ├── glue_shared_lib/
│       │   ├── Makefile
│       │   ├── setup.py
│       │   ├── src
│       │   │   └── glue_shared/__init__.py
│       └── tests/
└── terraform/
```

Let's describe the structure above.

- `/glue/` holds all the python code
- `/glue/data_sources/` holds the code of jobs for each data source
- `/glue/data_sources/ds1/` - is a directory of 1 particular data source containing transformation
- `/glue/data_sources/ds1/raw_to_refined` and `/glue/data_sources/ds1/refined_to_curated` are the transformations whose content is then deployed as a particular AWS Glue Job

- `/glue/shared/` - contains shared items among the glue (jobs, files, etc...)
- `/glue/shared/glue_shared_lib` - is the library used by the jobs, contains configuration interface and other useful functions
- `/terraform/` holds all resources required to be deployed our Glue Jobs, IAM roles, lambda functions, etc...

Now that we understand the structure, we can look closer at a particular job.

## Glue Job structure

This is a standard blueprint which fit my purpose of developing and deploying several AWS Glue jobs.

```
ds1/
├── raw_to_refined/
│   ├── Makefile
│   ├── config.py
│   ├── raw_to_refined.py
│   └── requirements.txt
```

In this case, we are looking at a transformation job from raw zone to refined zone.

- `Makefile` - contains several make targets which names are common across all jobs, `clean`, `package`, `test`, `upload-job`, `upload`, `deploy` then implementation of each target is job-specific.
  - `clean` - Cleans up the local temporary files.
  - `package` - For PySpark job creates a `.zip` file with dependencies. For Python shell job it runs `pip` and downloads all the wheel files.
  - `upload-job` - uploads entry point script to S3 - useful for quick updates during the development in case you are not doing any changes inside dependent files.
  - `upload` - upload all related files `.zip`, `.whl` and entrypoint `.py` file to S3.
  - `deploy` - performs `clean`, `package` and `upload`
- `config.py` - is responsible for creating a configuration object. This is an `extra .py` file which is later packaged and used as a dependency. For the sake of saving some time I used python dictionary but with growing complexity of the job I'd recommend spending time on creating a better approach.
- `raw_to_refined.py` - this is the main entry point file executed by AWS Glue. U can use this file execute the code in dependencies or directly implement transformation logic. The name of this file is purposely the same as it's parent directory which will be explained later.

- `requirements.txt` - Is a standard [requirements file](#) It's a very simple way of managing your dependencies.

This setup gives me enough flexibility as a developer to run and update jobs in the cloud from within my local environment as well as using CI/CD. Another benefit is that if you have PySpark with Glue running locally, you can use that as well!

## Terraform part

This is an example of deploying PySpark Job via Terraform, Python Shell job follows the same process with a slight difference (mentioned later).

To create or update the job via Terraform we need to supply several parameters Glue API which Terraform resource requires. Plus the parameters our job expects.

```
1  ds1_raw_to_refined_job_glue_version = "2.0"
2
3  ds1_raw_to_refined_job_default_arguments = {
4    "--extra-py-files"      = "s3://bucket/ds1/raw_to_refined/dependencies/ds1_raw_to_r
5    "--TempDir"             = "s3://aws-glue-temporary-<account_id>-us-east-1"
6    "--job-language"        = "python"
7    "--APP_ENVIRONMENT"     = "dev"
8    "--LOG_LEVEL"           = "DEBUG"
9  }
10
11 ds1_raw_to_refined_job_connections      = ["poc-report-dev"]
12 ds1_raw_to_refined_job_command          = "glueetl" // or "pythonshell"
13 ds1_raw_to_refined_job_python_version   = "3"
14 ds1_raw_to_refined_job_script_location  = "s3://bucket/ds1/refined_to_curated/refined_to_cu
```

tf1\_sample.tf hosted with ❤️ by GitHub

[view raw](#)

We need to provide:

- `Command.ScriptLocation` - represented as  
`ds1_raw_to_refined_job_script_location` - this is our entrypoint script
- `DefaultArguments` - represented as map  
`ds1_raw_to_refined_job_default_arguments` -- this holds the main configuration

**Key** `--extra-py-files` in the map `ds1_raw_to_refined_job_default_arguments` is a comma separated string of S3 urls pointing to our dependencies eg.

`s3://bucket/dep1.zip,s3://bucket/deb2.zip`

All extra dependencies fit in 1 `.zip` file and once you get the shape of these parameters there is no need to change it.

This brings a potential problem of human oversight, especially with Python Shell jobs. Where dependencies are wheels and by default, wheel name contains a version number `numpy-1.19.0-cp36-cp36m-manylinux2010_x86_64.whl`.

Then any change in `requirements.txt` or job arguments also requires a change in Terraform resource which is maintained in a different directory.


I haven't solved this problem during the project but this could be potentially avoided by maintaining a list of dependencies in the S3 bucket in the form of a file which could be generated during the `make`. And then Terraform would just download this information. However this theoretical the solution can lead to chicken-egg problems and I wish AWS Glue had a better option to maintain dependencies and the job config. Just allowing to use S3 prefix instead of the full URL would be a good start.


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 | <b>AWS Glue first experience - part 4 - Deployment &amp; packaging</b> |
| 5 | AWS Glue first experience - part 5 - Glue Workflow, monitoring an...   |

Top comments (1) ↕

 **Mauro Mascia** • Feb 8 '21 ⋮

 A note for the TempDir in Glue ETL Pyspark case: the fact that is temporary could probably let underestimate its role in the process. Instead, from my experience, it is

preferred to be pointed to a specific prefix in a bucket to avoid reaching the limit of 3,500 PUT/COPY/POST/DELETE or 5,500 GET/HEAD requests per second.

So it is better to use i.e. `s3://aws-glue-temporary-us-east-1/ds1_raw_to_refined/`, specifying a prefix for each job.

In fact an error like "503 Slow Down" may appear, without however indicating the source of the problem, which it can be the temporary folder, , not the destination one.

[Code of Conduct](#) • [Report abuse](#)

## Take a look at this:

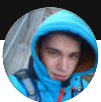
### Content

What is your approximate experience level (1-5)?

|             |               |                |               |             |
|-------------|---------------|----------------|---------------|-------------|
| 1<br>Novice | 2<br>Beginner | 3<br>Mid-level | 4<br>Advanced | 5<br>Expert |
|-------------|---------------|----------------|---------------|-------------|

This will not be displayed on your profile or anywhere publicly. It helps gently determine what content you are shown along with tags you follow etc.

Go to [your customization settings](#) to nudge your home feed to show content more relevant to your developer experience level. 🛠️



**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 3 - Arguments & Logging

[#aws](#) [#python](#) [#datascience](#) [#serverless](#)

---

AWS Glue first experience - part 2 - Dependencies and guts

[#aws](#) [#datascience](#) [#python](#) [#serverless](#)

---