Only you can see this message

This story's distribution setting is on. You're in the Partner Program, so this story is eligible to earn money. Learn more

Let's dive into setting up AWS Glue with an ETL job, then querying the data via AWS Athena

Craig Godden-Payne
Feb 7 · 5 min read ★

Let's look at a simple end to end run through of using AWS Glue to transform data from a format, into a more queryable format, and then query it using AWS Athena. We will look at this through the console only, with more focus on how to automate this with terraform in the future post.

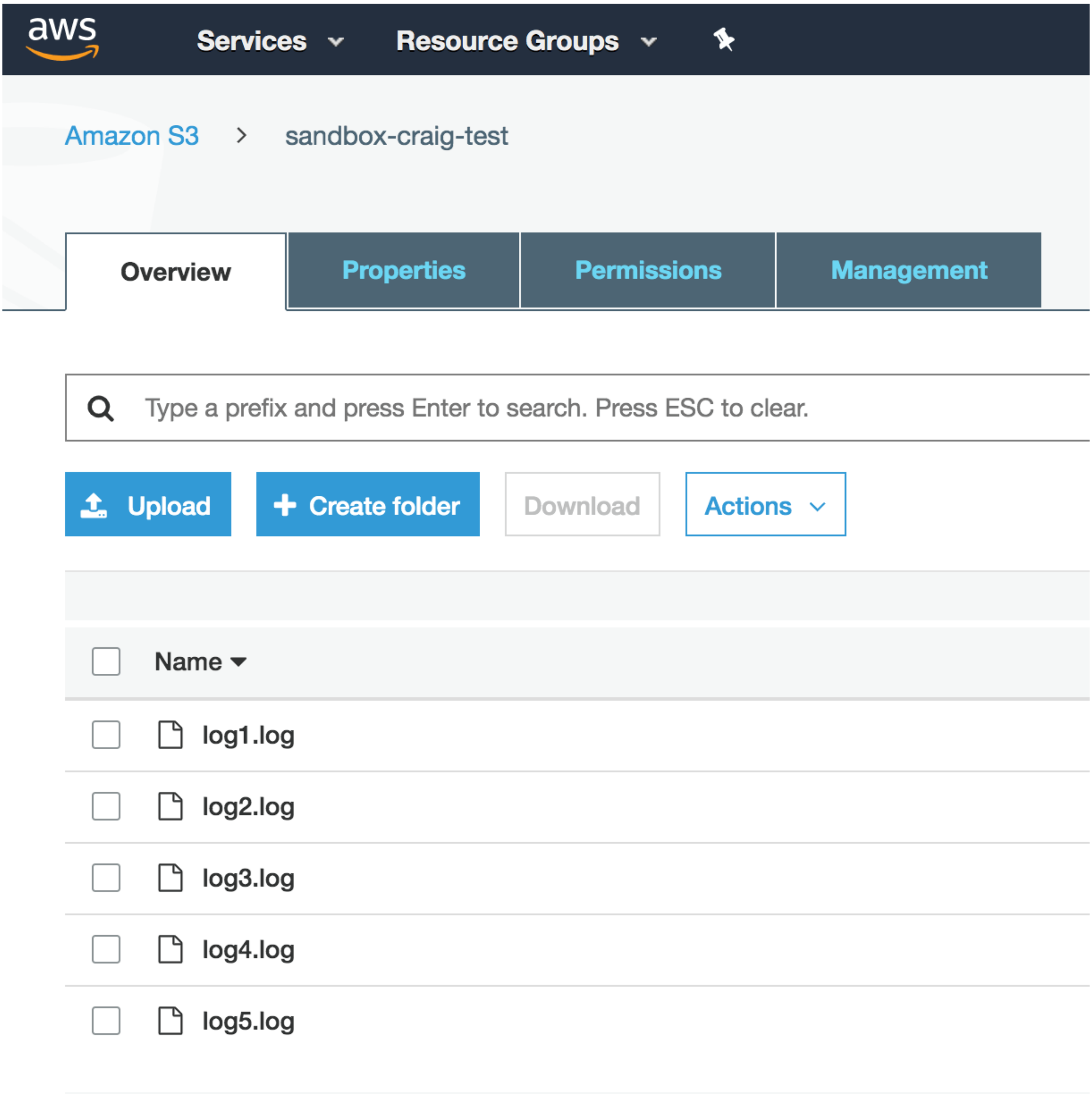
All the data in this post are from apache logs, which can be downloaded from Github. The data has been broken into 5 pieces, to simulate that the logs were uploaded at 5 different times.

You can download the splits here: log 1 log 2 log 3 log 4 log 5

. . .

Uploading the data

In order to query the data in AWS, you will need to upload the data files into an S3 bucket, you can use the example files above, or just some random apache log files.



□ log1.log
□ log2.log
□ log3.log
□ log4.log
□ log5.log

S3 bucket

. . .

Setting up an AWS Glue Job

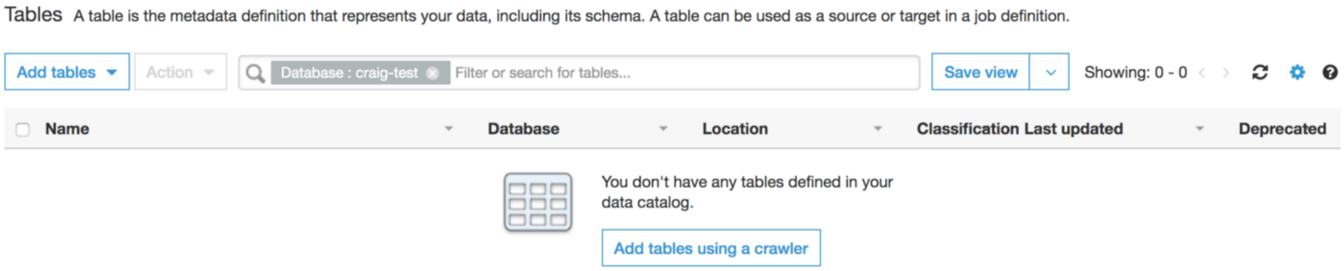
In the AWS console, search for Glue. Once it is open, navigate to the Databases tab. Create a new database, I created a database called craig-test. The database is used as a data catalog and stores information about schema information, not actual data.

Data Driven Investor | Microsoft Having An 'Edge' Over Chrome

A Brief History I was never a fan of browsers, well to be exact I was only a fan of one, Chrome. It has been my…

www.datadriveninvestor.com

Check the tables section, you will see the message that no tables have been defined in the data catalog. The next thing to do is to create a crawler, which will gather metadata about the data.



Tables A table is the metadata definition that represents your data, including its schema. A table can be used as a source or target in a job definition.

Name	Database	Location	Classification	Last updated	Deprecated

You don't have any tables defined in your data catalog.

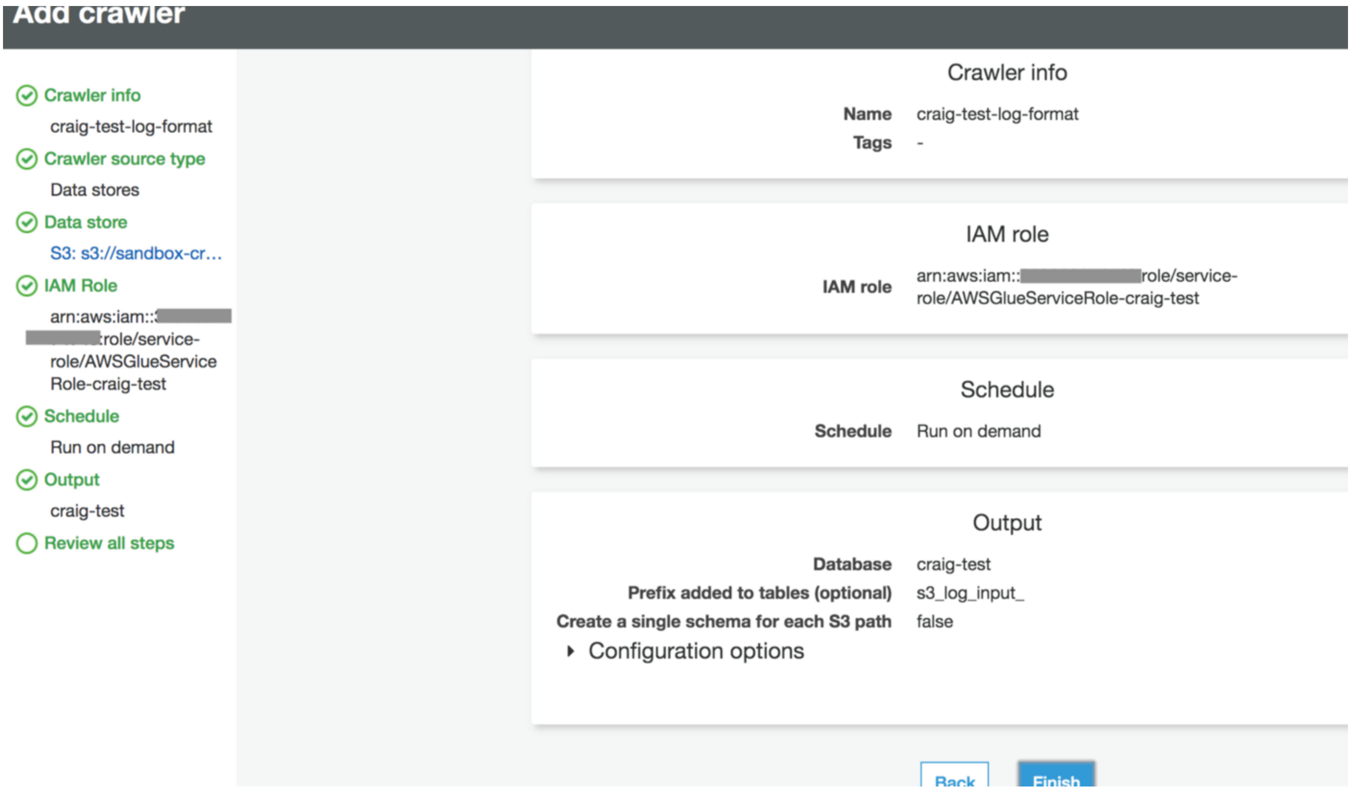
Data calatlog view

. . .

Creating a crawler

Whilst remaining in glue, create a crawler.

Point the crawler to look at the S3 bucket, and set it to write to the database data catalogue that was created before.



Add crawler

- Crawler info
  craig-test-log-format
- Crawler source type
  Data stores
- Data store
  S3: s3://sandbox-cr…
- IAM Role
  arn:aws:iam::[redacted]role/service-role/AWSGlueServiceRole-craig-test
- Schedule
  Run on demand
- Output
  craig-test
- Review all steps

Crawler info

Name craig-test-log-format
Tags -

IAM role

IAM role arn:aws:iam::[redacted]role/service-role/AWSGlueServiceRole-craig-test

Schedule

Schedule Run on demand

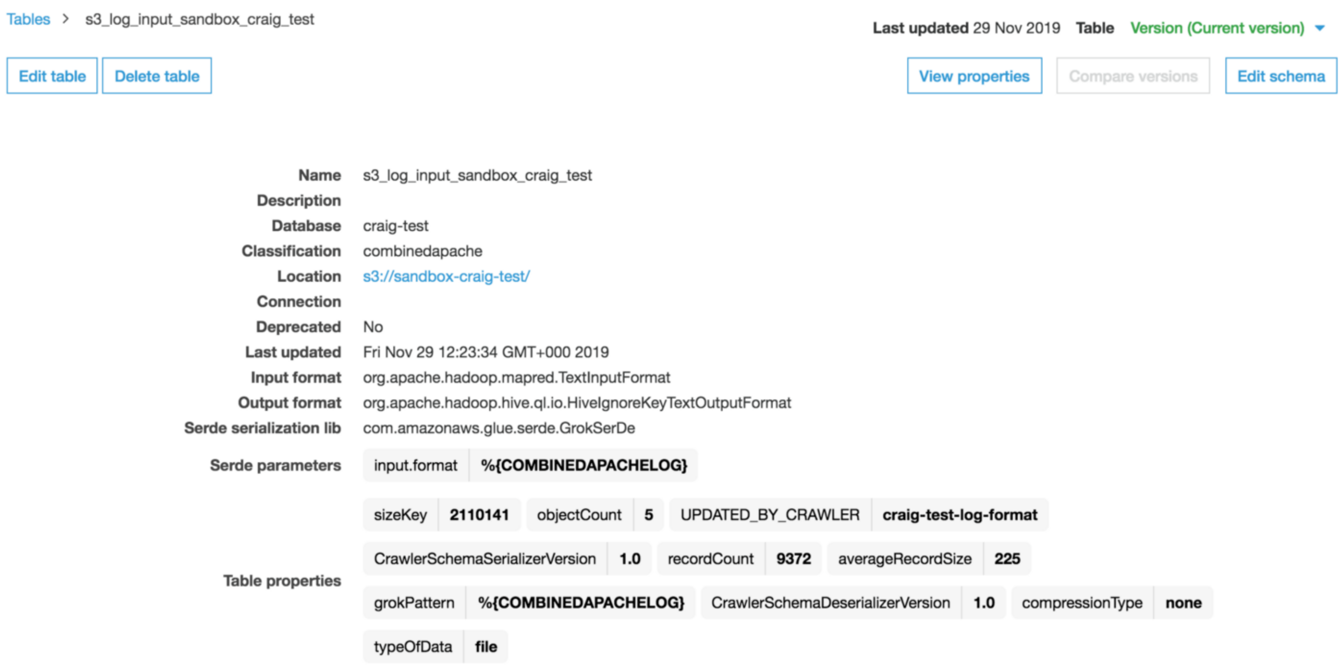
Output

Database craig-test
Prefix added to tables (optional) s3_log_input_
Create a single schema for each S3 path false
▸ Configuration options

Add a crawler

If you set the crawler to be on demand, you need to run it once you have finished creating the crawler.

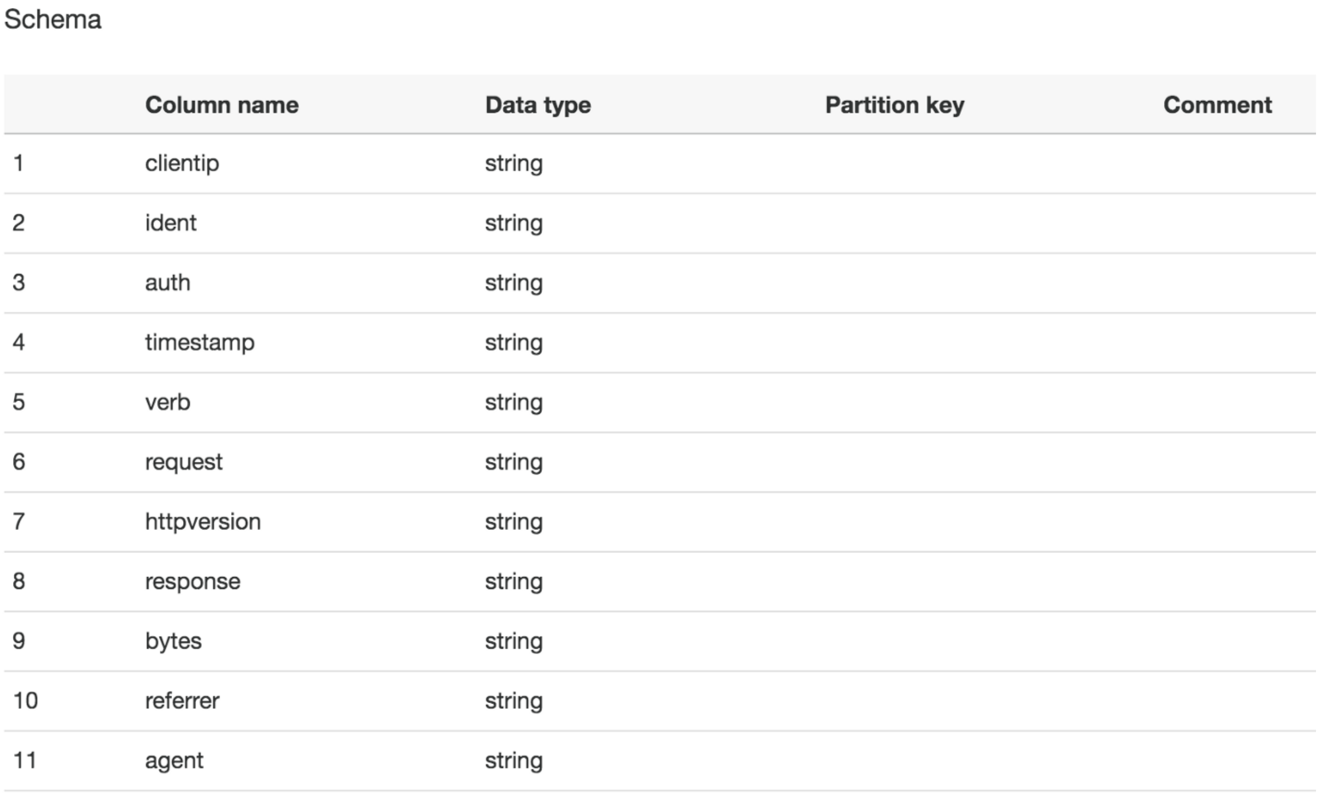
Glue will pretty pretty quickly crawl and determine the format of the log file. For my case, it mentions the data looks like it is in apache format, and it detailed what the schema looks like.



Last updated 29 Nov 2019 Table Version (Current version)

Name s3_log_input_sandbox_craig_test
Description
Database craig-test
Classification combinedapache
Location s3://sandbox-craig-test/
Connection
Deprecated No
Last updated Fri Nov 29 12:23:34 GMT+000 2019
Input format org.apache.hadoop.mapred.TextInputFormat
Output format org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Serde serialization lib com.amazonaws.glue.serde.GrokSerDe
Serde parameters input.format %{COMBINEDAPACHELOG}
Table properties sizeKey 2110141 objectCount 5 UPDATED_BY_CRAWLER craig-test-log-format CrawlerSchemaSerializerVersion 1.0 recordCount 9372 averageRecordSize 225 grokPattern %{COMBINEDAPACHELOG} CrawlerSchemaDeserializerVersion 1.0 compressionType none typeOfData file

Example of what the schema looks like



Schema

	Column name	Data type	Partition key	Comment
1	clientip	string		
2	ident	string		
3	auth	string		
4	timestamp	string		
5	verb	string		
6	request	string		
7	httpversion	string		
8	response	string		
9	bytes	string		
10	referrer	string		
11	agent	string		

More detail of the schema

• • •

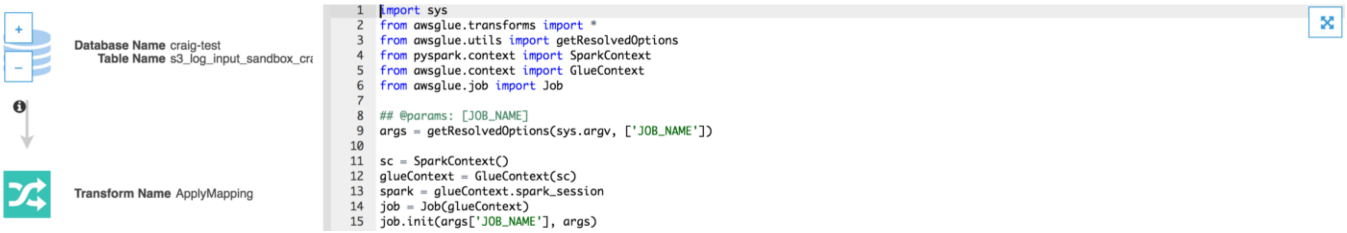
Creating the ETL job

The next stage is to create an ETL job, which will take the logs, and transform them into columnar data in parquet format, from apache log format.

It is much more efficient and cost effective to query in Athena using a format such as parquet rather than having to scan the whole data each time.

Created n Glue job, and add a data source, I used craig-test bucket. Set the destination bucket, I used craig-test-processed bucket.

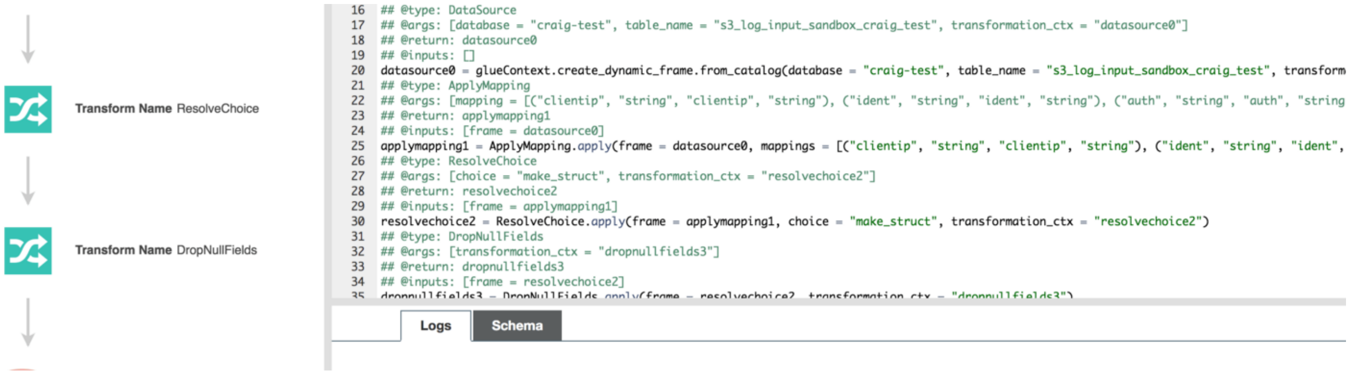
Select that you want to have the destination in parquet format. On clicking next, the apache spark code will be auto generated, but easy enough to read and make changes to if it needs to.



Database Name craig-test
Table Name s3_log_input_sandbox_cra

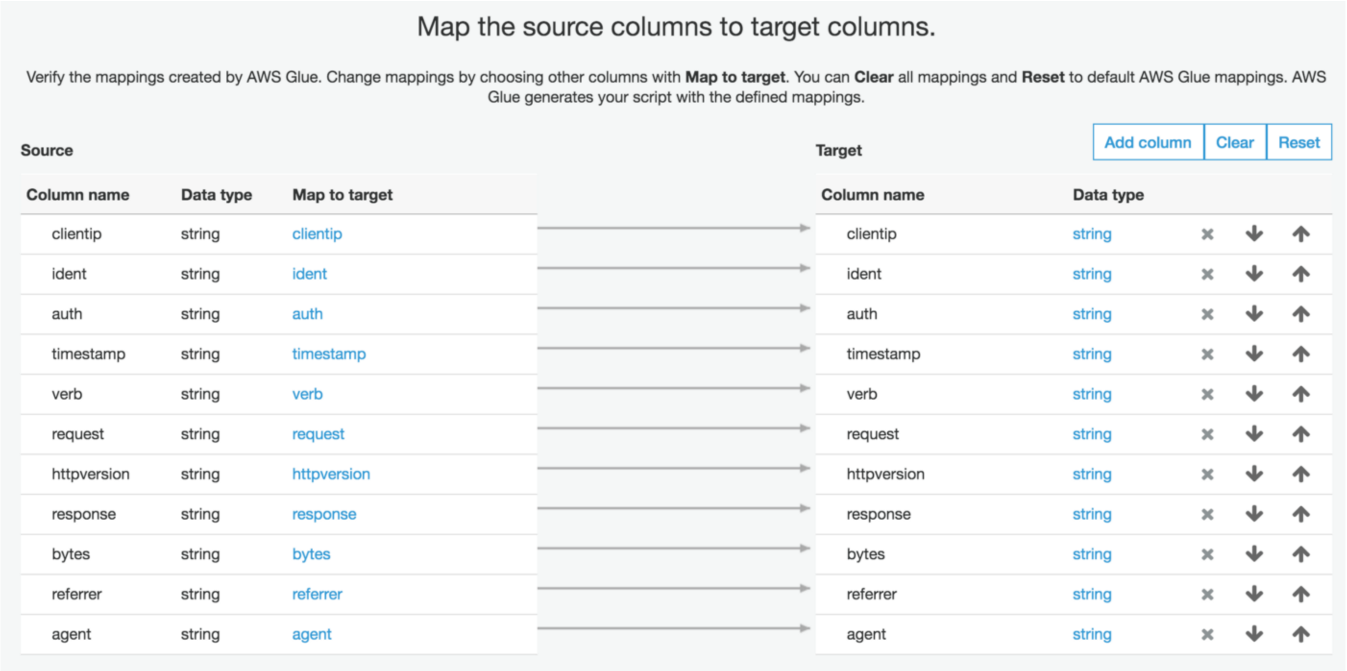
Transform Name ApplyMapping

```
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

## @params: [JOB_NAME]
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

sc = SparkContext()
glueContext = GlueContext(sc)
spark = glueContext.spark_session
job = Job(glueContext)
job.init(args['JOB_NAME'], args)
```



```
## @type: DataSource
## @args: [database = "craig-test", table_name = "s3_log_input_sandbox_craig_test", transformation_ctx = "datasource0"]
## @return: datasource0
## @inputs: []
datasource0 = glueContext.create_dynamic_frame.from_catalog(database = "craig-test", table_name = "s3_log_input_sandbox_craig_test", transform
## @type: ApplyMapping
## @args: [mapping = [("clientip", "string", "clientip", "string"), ("ident", "string", "ident", "string"), ("auth", "string", "auth", "string
## @return: applymapping1
## @inputs: [frame = datasource0]
applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("clientip", "string", "clientip", "string"), ("ident", "string", "ident",
## @type: ResolveChoice
## @args: [choice = "make_struct", transformation_ctx = "resolvechoice2"]
## @return: resolvechoice2
## @inputs: [frame = applymapping1]
resolvechoice2 = ResolveChoice.apply(frame = applymapping1, choice = "make_struct", transformation_ctx = "resolvechoice2")
## @type: DropNullFields
## @args: [transformation_ctx = "dropnullfields3"]
## @return: dropnullfields3
## @inputs: [frame = resolvechoice2]
dropnullfields3 = DropNullFields.apply(frame = resolvechoice2, transformation_ctx = "dropnullfields3")
```

Auto Generated Code



Map the source columns to target columns.

Verify the mappings created by AWS Glue. Change mappings by choosing other columns with **Map to target**. You can **Clear** all mappings and **Reset** to default AWS Glue mappings. AWS Glue generates your script with the defined mappings.

Source Column name	Source Data type	Map to target	Target Column name	Target Data type
clientip	string	clientip	clientip	string
ident	string	ident	ident	string
auth	string	auth	auth	string
timestamp	string	timestamp	timestamp	string
verb	string	verb	verb	string
request	string	request	request	string
httpversion	string	httpversion	httpversion	string
response	string	response	response	string
bytes	string	bytes	bytes	string
referrer	string	referrer	referrer	string
agent	string	agent	agent	string

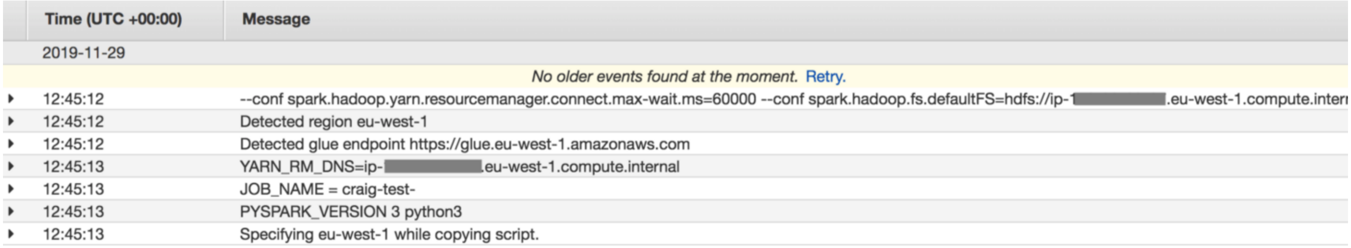
Auto mapped columns

Once you have completed creating a job, run the job. After a few minutes the job will be completed.

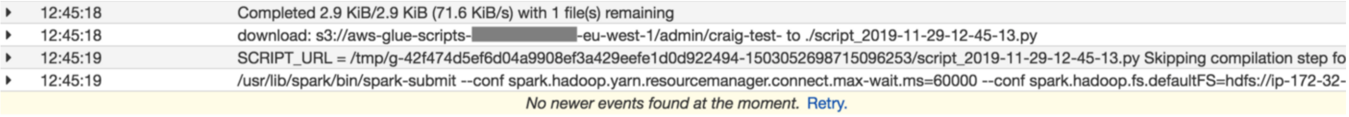
. . .

Issues or Failures

On my first run, I forgot about permissions, but if you hit any issues, they are pretty easy to diagnose by looking at Cloudwatch.



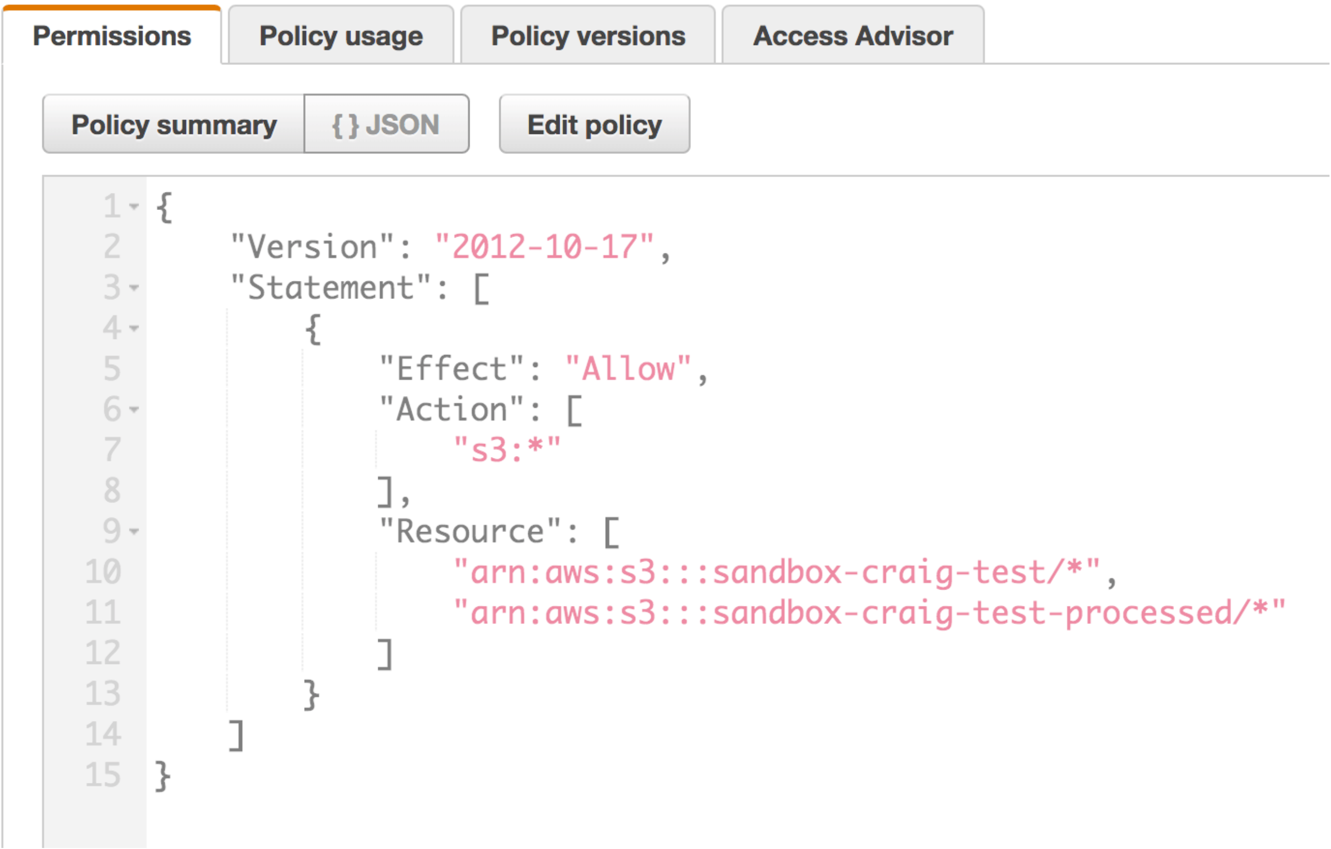
```
Time (UTC +00:00)   Message
2019-11-29
                    No older events found at the moment. Retry.
▸ 12:45:12          --conf spark.hadoop.yarn.resourcemanager.connect.max-wait.ms=60000 --conf spark.hadoop.fs.defaultFS=hdfs://ip-1[redacted].eu-west-1.compute.intern
▸ 12:45:12          Detected region eu-west-1
▸ 12:45:12          Detected glue endpoint https://glue.eu-west-1.amazonaws.com
▸ 12:45:13          YARN_RM_DNS=ip-[redacted].eu-west-1.compute.internal
▸ 12:45:13          JOB_NAME = craig-test-
▸ 12:45:13          PYSPARK_VERSION 3 python3
▸ 12:45:13          Specifying eu-west-1 while copying script.
```



```
▸  12:45:18   Completed 2.9 KiB/2.9 KiB (71.6 KiB/s) with 1 file(s) remaining
▸  12:45:18   download: s3://aws-glue-scripts-[redacted]-eu-west-1/admin/craig-test- to ./script_2019-11-29-12-45-13.py
▸  12:45:19   SCRIPT_URL = /tmp/g-42f474d5ef6d04a9908ef3a429eefe1d0d922494-1503052698715096253/script_2019-11-29-12-45-13.py Skipping compilation step fo
▸  12:45:19   /usr/lib/spark/bin/spark-submit --conf spark.hadoop.yarn.resourcemanager.connect.max-wait.ms=60000 --conf spark.hadoop.fs.defaultFS=hdfs://ip-172-32-
                                        No newer events found at the moment. Retry.
```

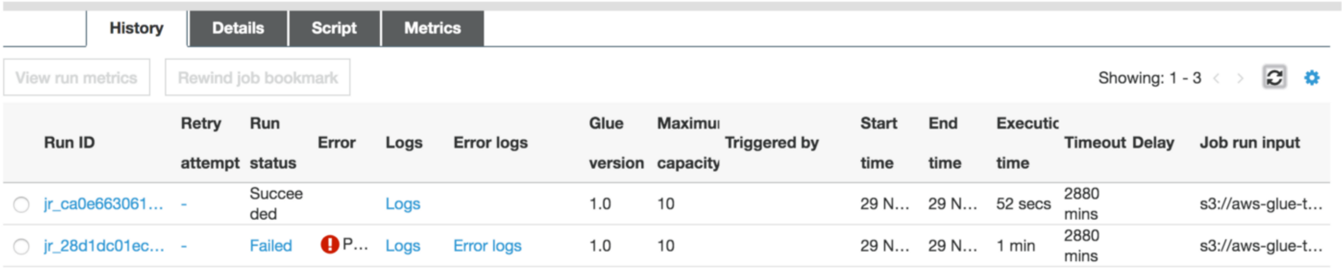
Cloudwatch

In my case, it was due to IAM permissions. I forgot to allow the glue role to have access to my output bucket.



```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:*"
            ],
            "Resource": [
                "arn:aws:s3:::sandbox-craig-test/*",
                "arn:aws:s3:::sandbox-craig-test-processed/*"
            ]
        }
    ]
}
```

Bucket policy



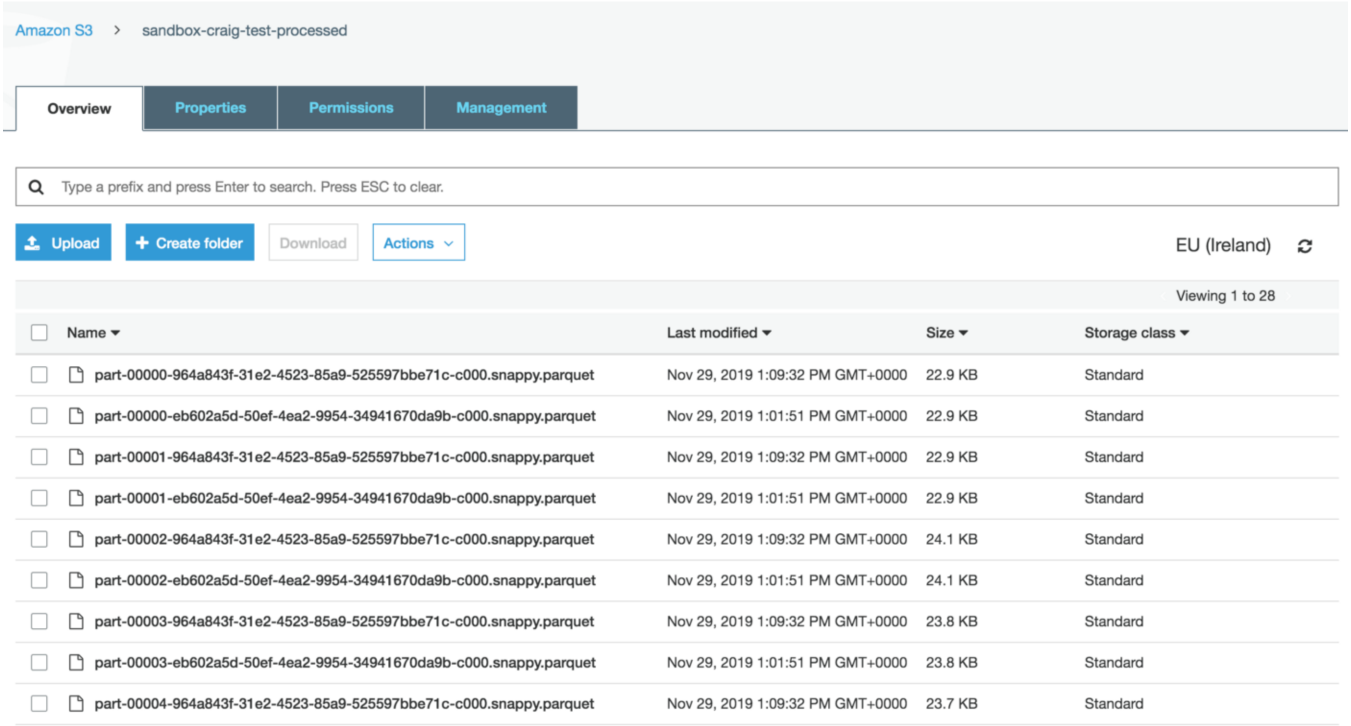
	Run ID	Retry attempt	Run status	Error	Logs	Error logs	Glue version	Maximu capacity	Triggered by	Start time	End time	Executic time	Timeout	Delay	Job run input
○	jr_ca0e663061...	-	Succeeded		Logs		1.0	10		29 N...	29 N...	52 secs	2880 mins		s3://aws-glue-t...
○	jr_28d1dc01ec...	-	Failed	❗P...	Logs	Error logs	1.0	10		29 N...	29 N...	1 min	2880 mins		s3://aws-glue-t...

Job runs

• • •

Rebuilding the schema

Check the s3 processed bucket, and you should find the parquet files are there ready for me to query using AWS Athena.

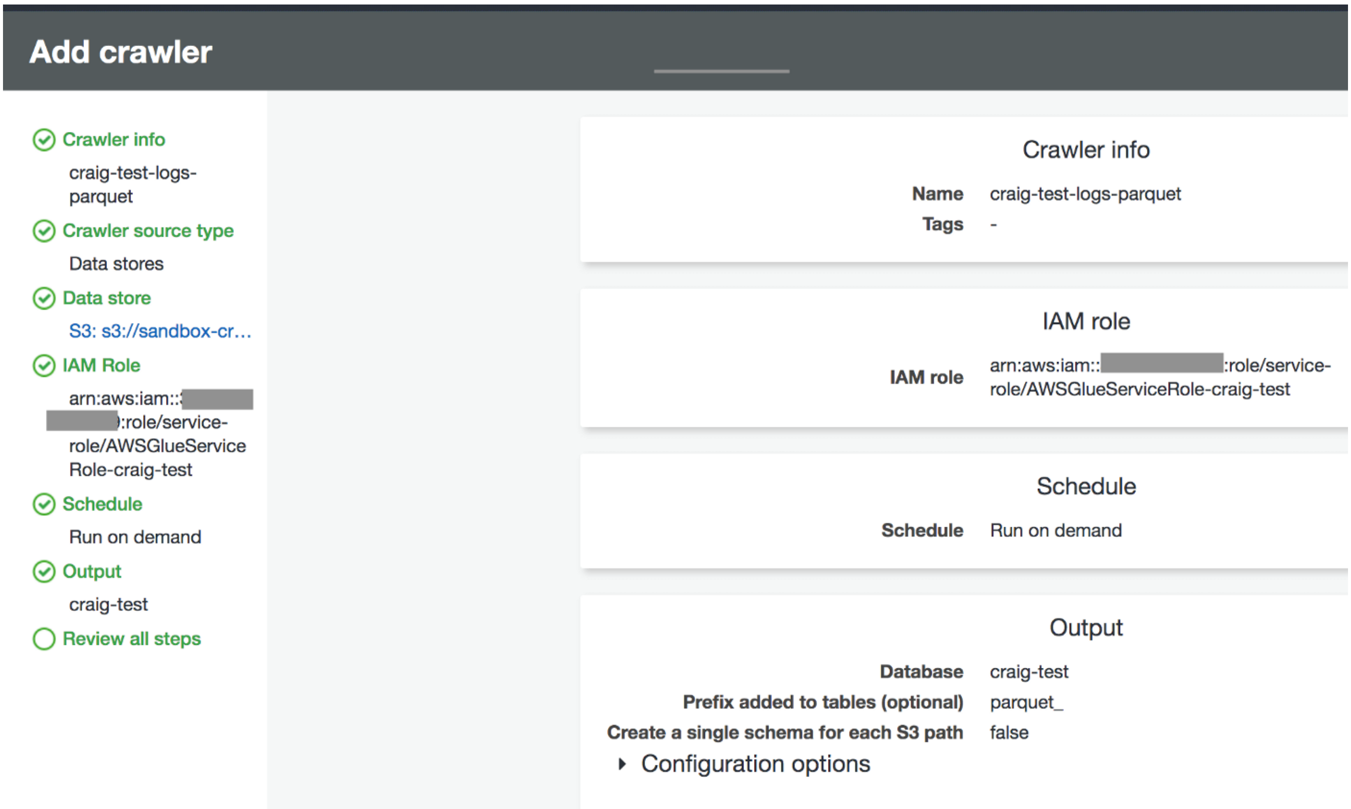


□	Name ▾	Last modified ▾	Size ▾	Storage class ▾
□	part-00000-964a843f-31e2-4523-85a9-525597bbe71c-c000.snappy.parquet	Nov 29, 2019 1:09:32 PM GMT+0000	22.9 KB	Standard
□	part-00000-eb602a5d-50ef-4ea2-9954-34941670da9b-c000.snappy.parquet	Nov 29, 2019 1:01:51 PM GMT+0000	22.9 KB	Standard
□	part-00001-964a843f-31e2-4523-85a9-525597bbe71c-c000.snappy.parquet	Nov 29, 2019 1:09:32 PM GMT+0000	22.9 KB	Standard
□	part-00001-eb602a5d-50ef-4ea2-9954-34941670da9b-c000.snappy.parquet	Nov 29, 2019 1:01:51 PM GMT+0000	22.9 KB	Standard
□	part-00002-964a843f-31e2-4523-85a9-525597bbe71c-c000.snappy.parquet	Nov 29, 2019 1:09:32 PM GMT+0000	24.1 KB	Standard
□	part-00002-eb602a5d-50ef-4ea2-9954-34941670da9b-c000.snappy.parquet	Nov 29, 2019 1:01:51 PM GMT+0000	24.1 KB	Standard
□	part-00003-964a843f-31e2-4523-85a9-525597bbe71c-c000.snappy.parquet	Nov 29, 2019 1:09:32 PM GMT+0000	23.8 KB	Standard
□	part-00003-eb602a5d-50ef-4ea2-9954-34941670da9b-c000.snappy.parquet	Nov 29, 2019 1:01:51 PM GMT+0000	23.8 KB	Standard
□	part-00004-964a843f-31e2-4523-85a9-525597bbe71c-c000.snappy.parquet	Nov 29, 2019 1:09:32 PM GMT+0000	23.7 KB	Standard

S3 output

You will need to rebuild the schema to include the new data sets. In my case I updated to include the transformed columnar parquet format.

This time, create a new crawler, and crawl the S3 bucket containing the processed files.



Add crawler

✓ Crawler info
craig-test-logs-parquet
✓ Crawler source type
Data stores
✓ Data store
S3: s3://sandbox-cr…
✓ IAM Role
arn:aws:iam::[redacted]:role/service-role/AWSGlueServiceRole-craig-test
✓ Schedule
Run on demand
✓ Output
craig-test
○ Review all steps

Crawler info

Name: craig-test-logs-parquet
Tags: -

IAM role

IAM role: arn:aws:iam::[redacted]:role/service-role/AWSGlueServiceRole-craig-test

Schedule

Schedule: Run on demand

Output

Database: craig-test
Prefix added to tables (optional): parquet_
Create a single schema for each S3 path: false

▸ Configuration options

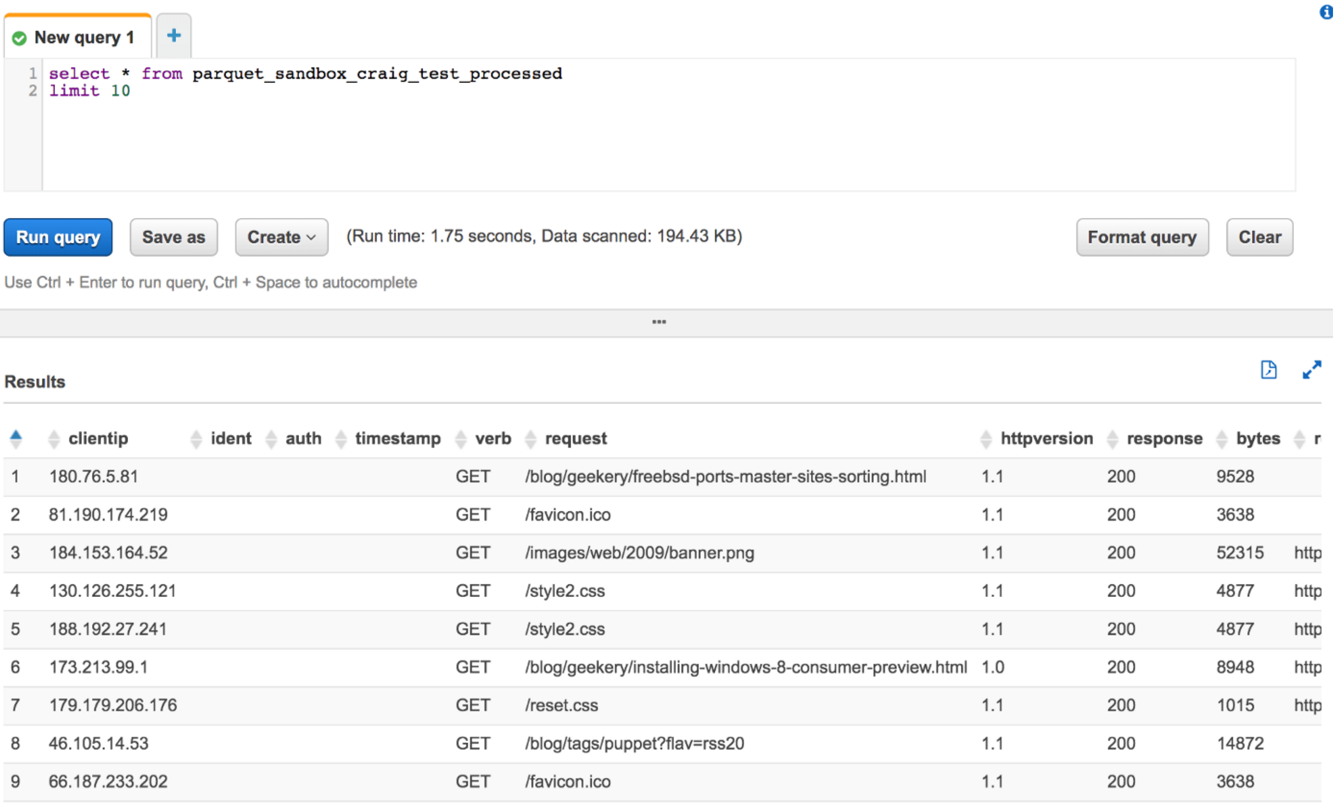
Add crawler

Once it has been ran, and the schema has been added to the data catalogue, it is time to move over to query.

. . .

Query using AWS Athena

Once in Athena, you can query the data from the logs. Querying the parquet logs is very cost effective and quick. whereas querying the original data is expensive and slow.



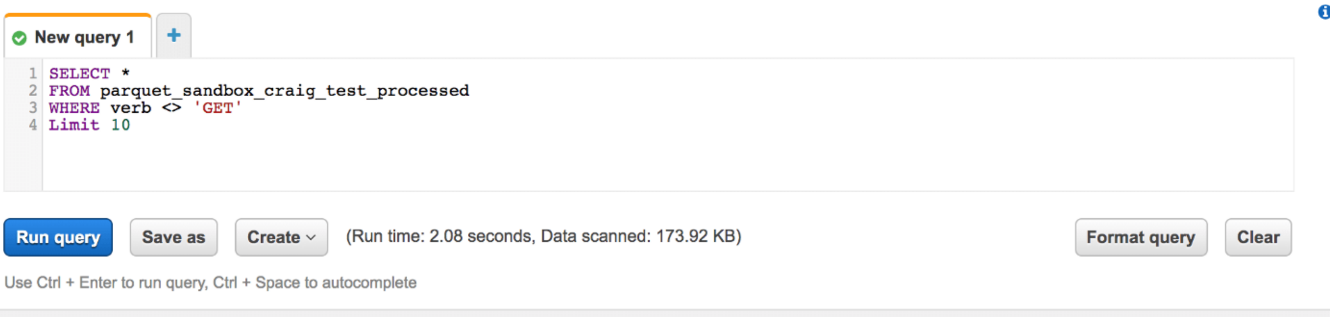
```
select * from parquet_sandbox_craig_test_processed
limit 10
```

(Run time: 1.75 seconds, Data scanned: 194.43 KB)

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

	clientip	ident	auth	timestamp	verb	request	httpversion	response	bytes	r
1	180.76.5.81				GET	/blog/geekery/freebsd-ports-master-sites-sorting.html	1.1	200	9528	
2	81.190.174.219				GET	/favicon.ico	1.1	200	3638	
3	184.153.164.52				GET	/images/web/2009/banner.png	1.1	200	52315	http
4	130.126.255.121				GET	/style2.css	1.1	200	4877	http
5	188.192.27.241				GET	/style2.css	1.1	200	4877	http
6	173.213.99.1				GET	/blog/geekery/installing-windows-8-consumer-preview.html	1.0	200	8948	http
7	179.179.206.176				GET	/reset.css	1.1	200	1015	http
8	46.105.14.53				GET	/blog/tags/puppet?flav=rss20	1.1	200	14872	
9	66.187.233.202				GET	/favicon.ico	1.1	200	3638	

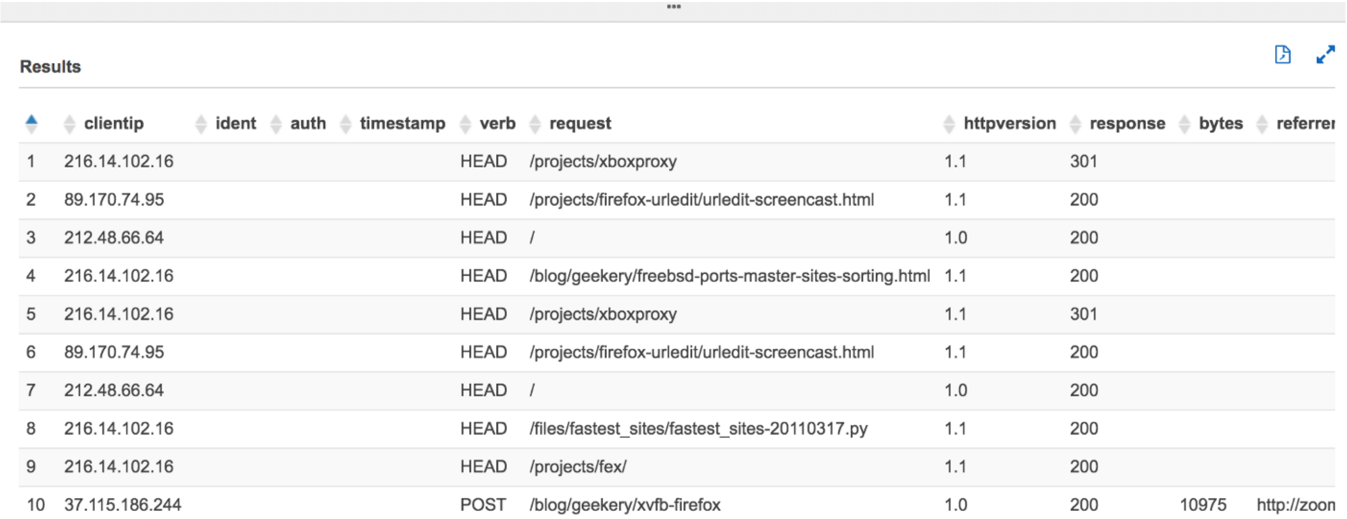
AWS Athena



```
SELECT *
FROM parquet_sandbox_craig_test_processed
WHERE verb <> 'GET'
Limit 10
```

(Run time: 2.08 seconds, Data scanned: 173.92 KB)

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

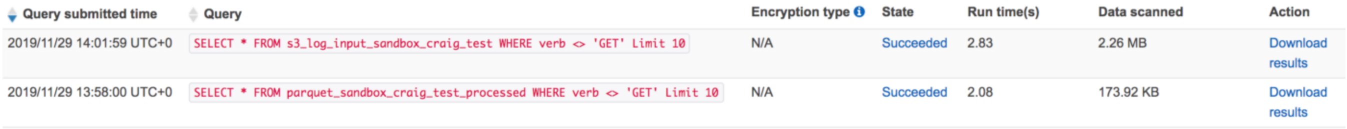


Results

	clientip	ident	auth	timestamp	verb	request	httpversion	response	bytes	referrer
1	216.14.102.16				HEAD	/projects/xboxproxy	1.1	301		
2	89.170.74.95				HEAD	/projects/firefox-urledit/urledit-screencast.html	1.1	200		
3	212.48.66.64				HEAD	/	1.0	200		
4	216.14.102.16				HEAD	/blog/geekery/freebsd-ports-master-sites-sorting.html	1.1	200		
5	216.14.102.16				HEAD	/projects/xboxproxy	1.1	301		
6	89.170.74.95				HEAD	/projects/firefox-urledit/urledit-screencast.html	1.1	200		
7	212.48.66.64				HEAD	/	1.0	200		
8	216.14.102.16				HEAD	/files/fastest_sites/fastest_sites-20110317.py	1.1	200		
9	216.14.102.16				HEAD	/projects/fex/	1.1	200		
10	37.115.186.244				POST	/blog/geekery/xvfb-firefox	1.0	200	10975	http://zoon

Athena Query

The difference in this simple query, is with the parquet data. If you look at the amount of data processed, you can see how much more expensive it would be



Query submitted time	Query	Encryption type	State	Run time(s)	Data scanned	Action
2019/11/29 14:01:59 UTC+0	`SELECT * FROM s3_log_input_sandbox_craig_test WHERE verb <> 'GET' Limit 10`	N/A	Succeeded	2.83	2.26 MB	Download results
2019/11/29 13:58:00 UTC+0	`SELECT * FROM parquet_sandbox_craig_test_processed WHERE verb <> 'GET' Limit 10`	N/A	Succeeded	2.08	173.92 KB	Download results

Conclusion

Wherever possible, query data in a structured format

Data Driven Investor

Gain Access to Expert Views

Email

First Name

Give me access!

□ I agree to leave Medium.com and submit this information, which will be collected and used according to Upscribe's privacy policy.

4.2K signups

AWS Glue Athena Python Data Analytics

About Help Legal

Get the Medium app

Download on the App Store

GET IT ON Google Play