CSV Analysis with Amazon Athena

Executing standard SQL queries on your Amazon S3 bucket files

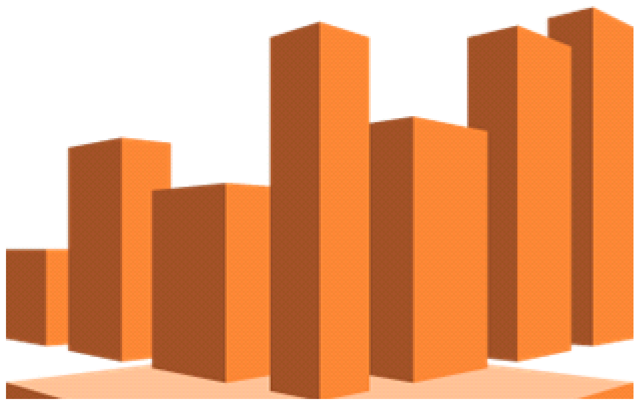
Ross Rhodes Follow
Dec 14, 2018 · 7 min read

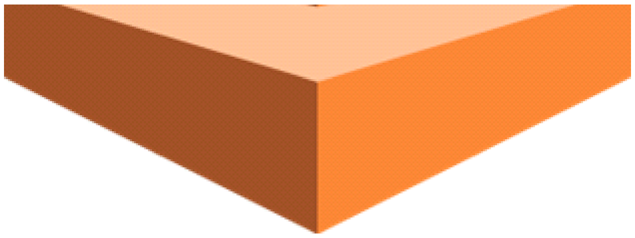
One of Amazon Web Services' many services for data management in the cloud, Amazon Athena allows us to query data we hold in another service called Amazon Simple Storage Service (S3) using standard SQL syntax.

S3 provides us with object-based storage, capable of storing flat files like plain-text, CSV, or JSON. These files are held in what we call buckets. What S3 doesn't support is block-based storage — the likes of databases or operating systems. Does this mean we would need to set up a database elsewhere in AWS in order to query the data in our files?

Before AWS re:Invent 2016, yes! You could set up a database in Amazon Relational Database Service (RDS) or Amazon DynamoDB, then define a function in AWS Lambda to transfer data from your S3 bucket to your database, and query your data from there.

Sounds a little over the top, right? Especially if you're new to AWS. With its general roll-out in November 2016, Athena saves us the trouble. Supporting CSV, JSON, and columnar data sets like Apache Parquet, Athena makes it a relatively painless task to analyse data in your S3 buckets.

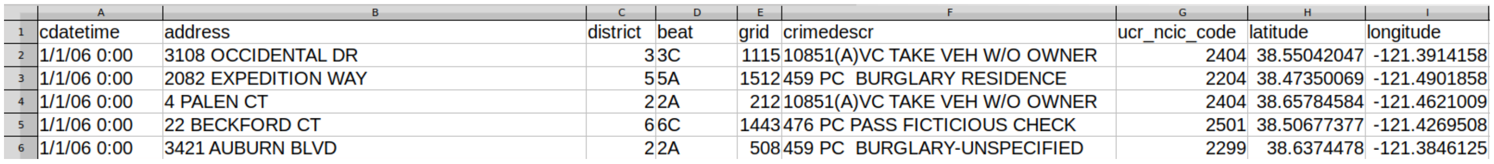



Credit: thirdeyedata.io

Working with a single S3 bucket, and a CSV data set acquired from SpatialKey Support, I wish to demonstrate how to use Athena to analyse CSV data with the Python programming language.

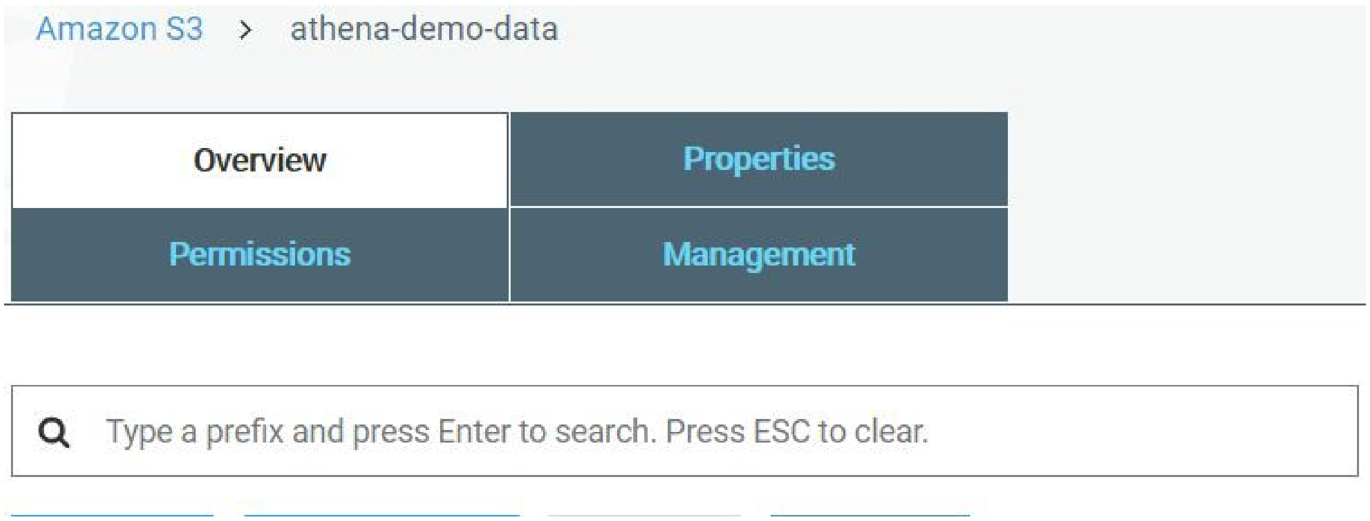
Background

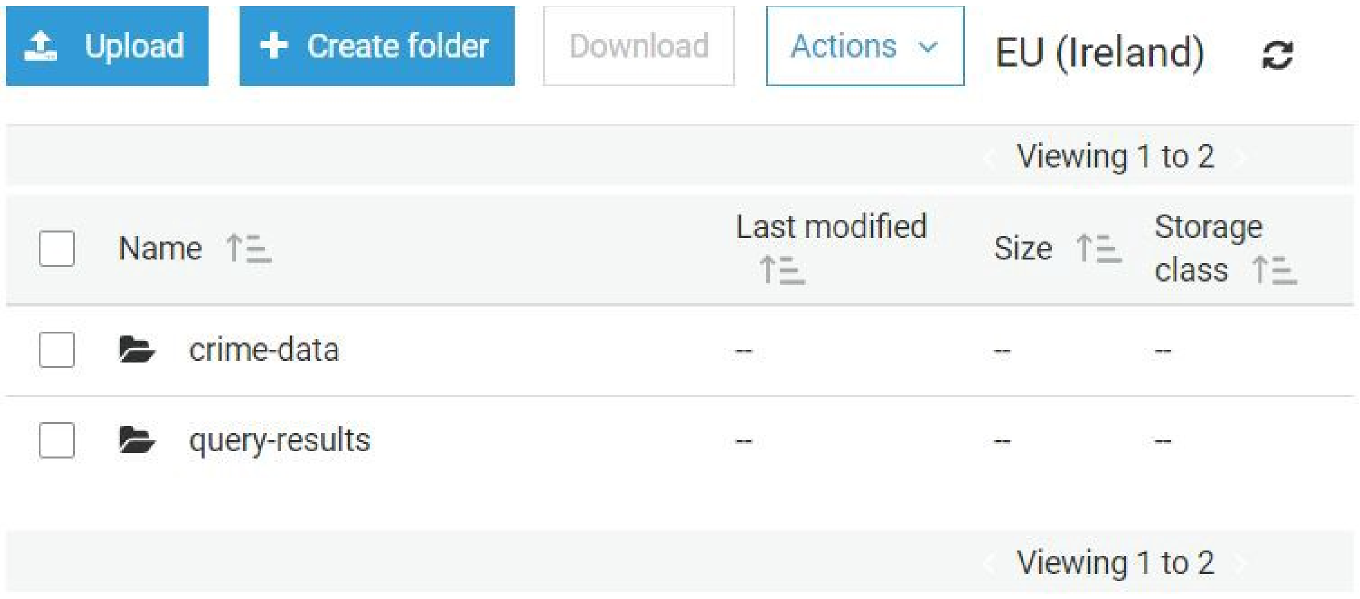
SpatialKey Support offers a range of sample CSV data sets, free to download. I've chosen their crime CSV file for this demo, made available by the Sacramento Police Department. This file records 7,584 crimes during January 2006 — a sufficient size for Athena to analyse.



	A	B	C	D	E	F	G	H	I
1	cdatetime	address	district	beat	grid	crimedescr	ucr_ncic_code	latitude	longitude
2	1/1/06 0:00	3108 OCCIDENTAL DR	3	3C	1115	10851(A)VC TAKE VEH W/O OWNER	2404	38.55042047	-121.3914158
3	1/1/06 0:00	2082 EXPEDITION WAY	5	5A	1512	459 PC BURGLARY RESIDENCE	2204	38.47350069	-121.4901858
4	1/1/06 0:00	4 PALEN CT	2	2A	212	10851(A)VC TAKE VEH W/O OWNER	2404	38.65784584	-121.4621009
5	1/1/06 0:00	22 BECKFORD CT	6	6C	1443	476 PC PASS FICTICIOUS CHECK	2501	38.50677377	-121.4269508
6	1/1/06 0:00	3421 AUBURN BLVD	2	2A	508	459 PC BURGLARY-UNSPECIFIED	2299	38.6374478	-121.3846125

We require a S3 bucket to hold this file in AWS. For the purposes of this demo, I've created a bucket by the name `athena-demo-data`. Inside this bucket are two folders: `crime-data` and `query-results`. The former holds our CSV file, whilst the latter — currently empty — will hold query results once Athena is up and running.


Amazon S3 > athena-demo-data
Overview
Properties
Permissions
Management
Type a prefix and press Enter to search. Press ESC to clear.



Name	Last modified	Size	Storage class
crime-data	--	--	--
query-results	--	--	--

Notice next to the “Actions” drop-down is the phrase “EU (Ireland)”. This phrase indicates which AWS region my S3 bucket resides in. This will be significant later when we come to configuring Athena, so keep that in mind!

With our CSV data in S3, we’re ready to configure Athena to execute some queries. Our tech stack for the job will consist of Python 3 and Amazon’s Python 3 client for AWS, Boto 3.

Configuration

First things first, we need to specify our requirements for this Python project. Luckily for us, we only have the one: Boto 3. Defined in `requirements.txt`, Boto 3 has been locked down to version 1.9.42, the default at the time of writing:

```
boto3==1.9.42
```

athena_requirements.txt hosted with ❤ by GitHub view raw

With Boto 3 installed, we need to instantiate a Boto 3 client for Athena. In the Python code, we pass in two parameters: the AWS service we wish to use, and the region in which to set up this client. This is where the region specified in our S3 bucket proves significant - make sure to select your bucket’s region, otherwise Athena will fail to find your S3 data. In this case, “EU (Ireland)” corresponds to region `eu-west-1`.

```
import boto3
```

```python

def main():
  athena_client = boto3.client('athena', region_name='eu-west-1')
```

athena_client.py hosted with ❤ by GitHub view raw

Querying the Data

With the Athena client ready, it's time to query your S3 data. There's five queries we'll execute in order to analyse the Sacramento Police Department's crime data:

1. create a database,
2. create a table within the database,
3. analyse the newly created table,
4. drop the table, and
5. drop the database.

Each query we execute through the Athena Boto 3 client requires a similar setup. Defined within my own `execute_query` method, we call the client's `start_query_execution` method, taking in a string SQL query as well as some result configurations.

```python
def execute_query(athena_client: boto3.client, query: str) -> dict:
    return athena_client.start_query_execution(
        QueryString=query,
        ResultConfiguration={
            "OutputLocation": "s3://athena-demo-data/query-results/",
            "EncryptionConfiguration": {
                "EncryptionOption": "SSE_S3"
            }
        }
    )
```

athena_query_execution.py hosted with ❤ by GitHub view raw

In this case, we take in two configurations. The first is our output location for query results, which is mandatory. I've set this to point to the `query-results` folder in my S3 bucket. The second is encryption configuration, which is optional. I pass in value `SSE-S3`, telling Amazon to encrypt the results and automatically rotate encryption keys on my behalf. There's alternative options available, outlined in the Boto 3 documentation for `start_query_execution`.

With my function in mind, we should explore the queries we will be passing into it. Creating the database — which I've called `sacramento` — is relatively straightforward using standard SQL:

```
CREATE DATABASE IF NOT EXISTS sacramento;
```

create_sacramento_db.sql hosted with ❤ by GitHub view raw

The next step, creating the table, is more interesting: not only does Athena create the table, but it also learns where and how to read the data from my S3 bucket. There's plenty of work going on behind the hood on this one. I've decided to call the table `crime_data`:

```
CREATE EXTERNAL TABLE IF NOT EXISTS sacramento.crime_data (
               ts VARCHAR(255),
               address VARCHAR(255),
               district INT,
               beat VARCHAR(255),
               grid INT,
               crimedescr VARCHAR(255),
               ucr_ncic_code INT,
               latitude FLOAT,
               longitude FLOAT
               )
               ROW FORMAT DELIMITED
               FIELDS TERMINATED BY ","
               LINES TERMINATED BY "\n"
               LOCATION 's3://athena-demo-data/crime-data/'
               TBLPROPERTIES (
               'skip.header.line.count' = '1'
               );
```

create_sacramento_table.sql hosted with ❤ by GitHub view raw

The `EXTERNAL` keyword on line one dictates that our data is originating from a source outwith the database, specified by the `LOCATION` parameter on line 15. This points to the `crime-data` folder from my S3 bucket. To assist Athena with its data parsing, we indicate how lines are separated, and how values are separated within a line, using the `LINES DELIMITED BY` and `FIELDS DELIMITED BY` phrases, respectively. Furthermore, we exclude the first line of our CSV file — consisting of header values — using `TBLPROPERTIES` on lines 16–18.

Each attribute from lines 2–10 corresponds to one column in the CSV file, with the first, `ts`, being timestamps of form “MM/DD/YY HH:MI”. In cases where there are leading zeroes in MM, DD, or HH, these are omitted. In other words, 1am on January 21st would look like 1/21/06 1:00, whilst 10pm on January 21st would like like 1/21/06 22:00. Given that all data for this file is from January, all the MMs are “1”!

With both database `sacramento` and table `crime_data` created, it’s time to analyse the latter to our heart’s content with some SQL `SELECT` statements. For the sake of demonstrating how this works, let’s write a query to fetch all crimes reported between January 1st and January 2nd in 2006.

As ever with programming, timestamps and dates prove interesting to work with. Critical to dealing with this data is knowing which query engine Athena uses under the hood.

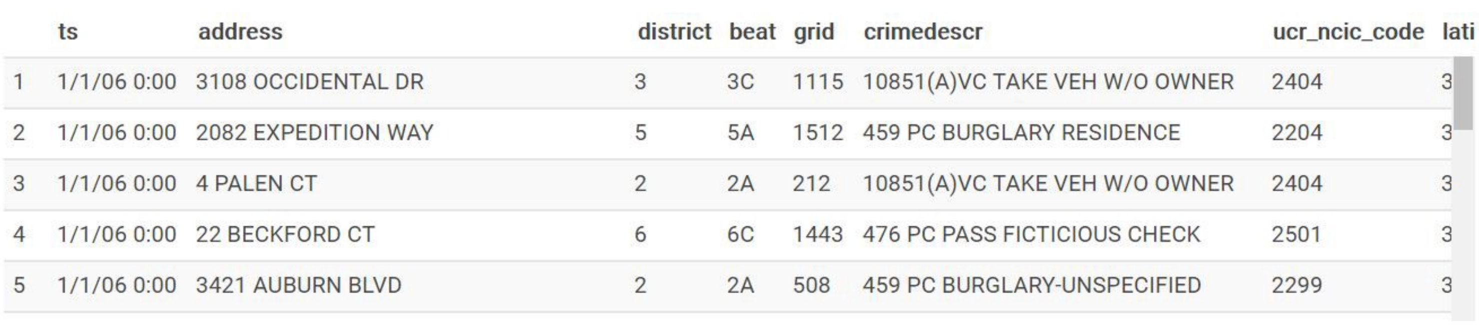
The answer? Presto, an open-source distributed query engine. Following Presto’s documentation on dates and times, we find ourselves parsing the provided timestamps with format specifiers in the form of `%c/%e/%y %k:%i` to make the following query:

```sql
SELECT *
FROM sacramento.crime_data
WHERE date_parse(ts, '%c/%e/%y %k:%i') >= timestamp '2006-01-01'
AND date_parse(ts, '%c/%e/%y %k:%i') <= timestamp '2006-01-02';
```

analyse_sacramento_table.sql hosted with ❤ by GitHub view raw

Running this query produced the expected results within a couple of seconds:

Results



	ts	address	district	beat	grid	crimedescr	ucr_ncic_code	lati
1	1/1/06 0:00	3108 OCCIDENTAL DR	3	3C	1115	10851(A)VC TAKE VEH W/O OWNER	2404	3
2	1/1/06 0:00	2082 EXPEDITION WAY	5	5A	1512	459 PC BURGLARY RESIDENCE	2204	3
3	1/1/06 0:00	4 PALEN CT	2	2A	212	10851(A)VC TAKE VEH W/O OWNER	2404	3
4	1/1/06 0:00	22 BECKFORD CT	6	6C	1443	476 PC PASS FICTICIOUS CHECK	2501	3
5	1/1/06 0:00	3421 AUBURN BLVD	2	2A	508	459 PC BURGLARY-UNSPECIFIED	2299	3

With this result set at our disposal, we should drop both our table and our database to wrap up our work. For each, a simple SQL `DROP` statement suffices:

```sql
DROP TABLE sacramento.crime_data;
```

drop_sacramento_table.sql hosted with ❤ by **GitHub** view raw

```sql
DROP DATABASE sacramento;
```

drop_sacramento_db.sql hosted with ❤ by **GitHub** view raw

Pricing

Whenever there's a bill coming our way, pricing lurks at the back of our minds. AWS is no exception, so it would be worth briefly explaining Athena's price plan before we call it a day.

Firstly, Athena does not charge for executing data definition language (DDL) queries — which includes our `CREATE` and `DROP` queries. This reduces our concerns from five queries to one.

For every terabyte of data scanned, Athena charges $5 (~£4 in my case). Athena excludes failed queries, but will include any data scanned from cancelled queries.

Given that the crime CSV file is only 775KB in size, I would need to execute our `SELECT` query over 2,580 times before I'm charged even a cent. Some might say its pricing is criminal.

If you're working with a larger data set and would like to keep costs low, consider converting your data into a columnar format such as Apache Parquet. In doing this, you may save yourself up to 90% compared to the original costs.

Conclusion

And that's us! Amazon Athena has analysed the Sacramento Police Department's crime data, reporting back to us crimes that were recorded between January 1st and January 2nd of 2006. That wasn't so bad, was it?

We've covered a fair amount here, particularly if AWS is completely new to you. If you would like to learn all things AWS, and fancy some pointers on where to begin some serious studying, then I would advise working toward AWS Solutions Architect — Associate certification following the study process outlined in my recent blog post. Whilst Athena is not covered in that particular course, many fundamental services — including S3 — are explained in great detail.

If you fancy playing around with Athena, you're more than welcome to fork my repository on GitLab. Since my bucket and security credentials remain private, you will need to configure a few things to get up and running.

First, you'll require an AWS account with MFA enabled. Assuming that to be set up, you will need to configure AWS CLI on your machine, and create an `AWS_MFA_SERIAL_NUMBER` environment variable with your MFA serial number, which can be found under your account security credentials.

In addition to that, you will need to create a S3 bucket to store your data. All names and file paths for your S3 bucket can be tweaked in the GitLab code through `config.py`, whilst queries can be altered in `queries.py`.

If you fancy digging further into what Boto 3 has to offer, their official documentation serves as a great resource to play around with Athena and other services offered through AWS.

. . .

For any questions, contact AVM Consulting: blog@avmconsulting.net. For further details on what AVM Consulting could do for you, visit their Well Architected Review page.

AWS Amazon Web Services Amazon Athena Software Development Software Engineering