AWS Database Blog

How to perform advanced analytics and build visualizations of your Amazon DynamoDB data by using Amazon Athena

by Roger Dahlstrom and Ilya Epshteyn | on 21 MAY 2018 | in Amazon Athena, Amazon DynamoDB, Analytics, Database | Permalink | Comments | Share

You can reap huge analytical value from billions of items and millions of requests per second in your Amazon DynamoDB service. However, you need to export your data in order to get that analytical value. Copying the data from a DynamoDB table to an analytics platform allows you to extract rich insights. In order to accomplish this, we find that a well-architected, big data pipeline helps you separate transactional processing from analytics. This blog post shows you how to build a big data pipeline that transitions the data from your DynamoDB table to Amazon S3. This helps you perform advanced analytics by using Amazon Athena, a fully managed Presto query service, and also helps you build visualizations and ad hoc analyses by using Amazon QuickSight.

Most decoupled big data applications have a common pipeline that separates storage from compute, which allows you to take advantage of new processing technologies as they arrive. Decoupling enables elastic provisioning of compute resources for multiple analytics engines without affecting the durability of the data. You might also want to design your pipeline so that the storage and processing stages repeat themselves to shape the data in a format that downstream applications can consume rapidly.

Three main characteristics influence the design of a big data pipeline:

- **Latency of the overall pipeline** – How much time do you have to go from data to insights? Milliseconds, minutes, or days?
- **Throughput of the data** – How much data needs to be ingested and processed? Are you dealing with gigabytes, terabytes, or petabytes?
- **Cost** – What is the target budget for your application? The most cost-effective option in AWS is often the right choice.

Other key considerations when designing your big data pipeline include data structure, access patterns, the temperature of the data, availability and durability, and whether the service is fully managed. Using the right tools for the job based on these characteristics is key to a well-architected big data pipeline.

Tiered big data pipeline

Before we dive into the tiered big data pipeline, let's review the key services and features that this solution uses.

Features of DynamoDB in your pipeline

In DynamoDB, tables, items, and attributes are the core components that you work with. A table is a collection of items, and each item is a collection of attributes. DynamoDB uses primary keys to identify each item in a table. Secondary indexes provide more querying flexibility. For more information, see Amazon DynamoDB: How It Works in the *DynamoDB Developer Guide.*

DynamoDB Time To Live (TTL) allows you to delete items automatically that are no longer relevant as a way to reduce storage costs. For this blog post, you enable TTL on the table and use the ttl attribute to set a timestamp for deletion. For more information about TTL, see Time to Live: How It Works.

You then use DynamoDB Streams to capture a time-ordered sequence of item-level modifications, which then you can copy to Amazon S3. For more information about DynamoDB Streams, see Capturing Table Activity with DynamoDB Streams.

Features of AWS Lambda in your pipeline

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. Lambda runs your code only when needed and scales automatically to thousands of requests per second. Lambda takes care of high availability, server and operating-system maintenance, and patching. With Lambda, you pay for only the consumed compute time when your code is running.

Lambda provides the ability to invoke your code in response to events, such as when an object is put into an S3 bucket. In our case, Lambda does so in response to updates made to the DynamoDB table. For this post, you create a Lambda function triggered by DynamoDB Streams to copy the items to Amazon S3 by using Amazon Kinesis Data Firehose.

You can copy all items to Amazon S3, if for example you want to perform analytics on the entire dataset. You can also copy only the items deleted by TTL. The records for items that are deleted by TTL contain an additional metadata attribute to distinguish them from items that are deleted by a user. The `userIdentity` field for TTL deletions (shown in the following example) indicates that DynamoDB performed the deletion as part of TTL.

```
"userIdentity": {
                "Type": "Service",
                "PrincipalId": "dynamodb.amazonaws.com"
            }
```

To transition only TTL data, modify the Lambda function to copy only the records where `eventName` is `REMOVE` and `userIdentity` contains `PrincipalID` equal to `dynamodb.amazonaws.com`.

Having Lambda as part of the pipeline provides an additional opportunity to perform light transformation on the data before storing it in Amazon S3. The Lambda function in this solution flattens some of the nested JSON data, which can potentially make it easier to join it with other datasets in the future.

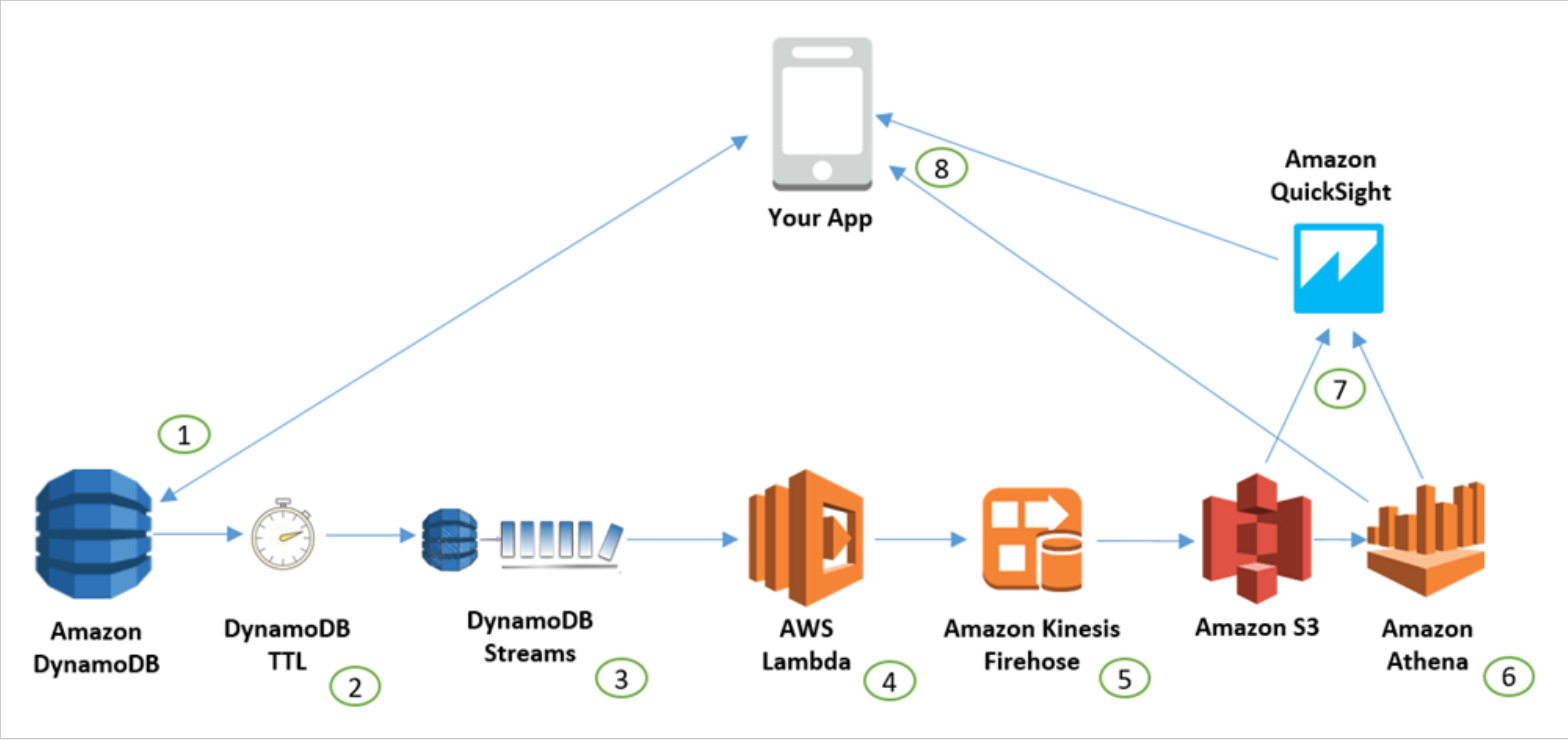
Features of Amazon Athena in your pipeline

Athena is an interactive query service that makes it easy to analyze data directly in Amazon S3 by using standard SQL. Athena is serverless, so there is no infrastructure to set up or manage, and you pay only for the queries you run. Athena scales automatically—executing queries in parallel—so results are fast, even with large datasets and complex queries. For more information about setting up Athena, see Setting Up in the *Athena User Guide.*

The data from DynamoDB lands in Amazon S3 in JSON format. Typically, we need an extract, transform, and load (ETL) process to convert the data into a format that is better suited for SQL queries. However, Athena uses an approach known as schema-on-read, which allows you to project your schema onto your data at the time you execute a query. This eliminates the need for any data loading or ETL. The solution in this post uses a JSON serializer/deserializer (SerDe) to parse the raw JSON records and create an external table using Hive data definition language. After the schema is created, you can begin querying the data.

Putting it all together

With all of the pieces explained, let's review the end-to-end pipeline. The following diagram illustrates how the solution works.



1. Your application writes data to a DynamoDB table. For the purposes of this blog post, we provide a sample Python function to populate the DynamoDB table by using a sample dataset.
2. The DynamoDB TTL table configuration expires and deletes items based on a time stamp attribute in the table.
3. The DynamoDB stream captures a time-ordered sequence of item-level modifications.
4. A Lambda function listens to the DynamoDB stream and writes the items to a Kinesis Data Firehose delivery stream. The provided function puts all new items to the Kinesis Data Firehose delivery stream. You can modify the function to put only records for items that are deleted by TTL based on the additional metadata attribute in the `userIdentity` field.
5. Kinesis Data Firehose sends the data to Amazon S3.
6. After the data is in Amazon S3, use Athena to create an external table, set up partitions, and begin querying the data.

7. You can also set up Amazon QuickSight to visualize the data and perform ad hoc queries of data in Athena or Amazon S3 directly.
8. Your application can query hot data directly from DynamoDB and also access analytical data through Athena APIs or Amazon QuickSight visualizations.

Deploying the solution

The provided AWS CloudFormation template deploys the DynamoDB table, DynamoDB stream, S3 bucket, Kinesis Data Firehose delivery stream, and the Lambda function. You need to manually configure Athena and Amazon QuickSight as described in the next section, "Validating the solution and querying the data."

To deploy the solution:

1. Download the Lambda function code ddb-to-firehose.py, and upload it to an S3 bucket of your choice. Make sure that the S3 bucket is in the same AWS Region where you plan to deploy the solution. Make note of the bucket name.
2. Download the CloudFormation template from GitHub to your desktop.
3. Navigate to the CloudFormation console and choose **CreateStack**.
4. Choose **Upload a template to Amazon S3** and browse to the `ddbathenablog_cf.yaml` template you have just downloaded. Choose **Next**.
5. Specify a stack name and DynamoDB table name (such as `Movies`). Also specify the bucket name where you uploaded the Lambda function code in Step 1. Choose **Next**.
6. Choose **Next** on the **Options**
7. On the final page, choose the **I acknowledge that AWS CloudFormation might create IAM resources** check box and choose **Create**.
8. Wait a few minutes until the AWS CloudFormation stack (a collection of AWS resources) is fully deployed and the status is shown as `CREATE_COMPLETE` before moving on to the next section.

Validating the solution and querying the data

To help validate the solution and quickly populate the DynamoDB table, download the data file and Python function linked in this post. The Python script loads the sample data file, which contains information about a few thousand movies from the Internet Movie Database (IMDb). For each movie, the file has a year, a title, and a JSON map named `info`. The following code example shows how the data looks.

```
{
    "year" : 2013,
    "title" : "Turn It Down, Or Else!",
    "info" : {
        "directors" : [
            "Alice Smith",
            "Bob Jones"
        ],
        "release_date" : "2013-01-18T00:00:00Z",
        "rating" : 6.2,
        "genres" : [
            "Comedy",
```

```
            "Drama"
        ],
        "image_url" : "http://ia.media-imdb.com/images/N/O9ERWAU7FS797AJ7LU8HN09AMUP908RLlo5JF90EWR7
        "plot" : "A rock band plays their music at high volumes, annoying the neighbors.",
        "rank" : 11,
        "running_time_secs" : 5215,
        "actors" : [
            "David Matthewman",
            "Ann Thomas",
            "Jonathan G. Neff"
        ]
    }
}
```

To validate the solution and populate the DynamoDB table:

1. Download the LoadMovieData.py script and the moviedata.zip. Extract the `moviedata.json` file from the `moviedata.zip` archive.
2. Edit the LoadMovieData.py script and update the values for `region_name` and `dynamodb`. `Table` to match your AWS Region and table name.

   ```
   dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
   table = dynamodb.Table('movies')
   ```

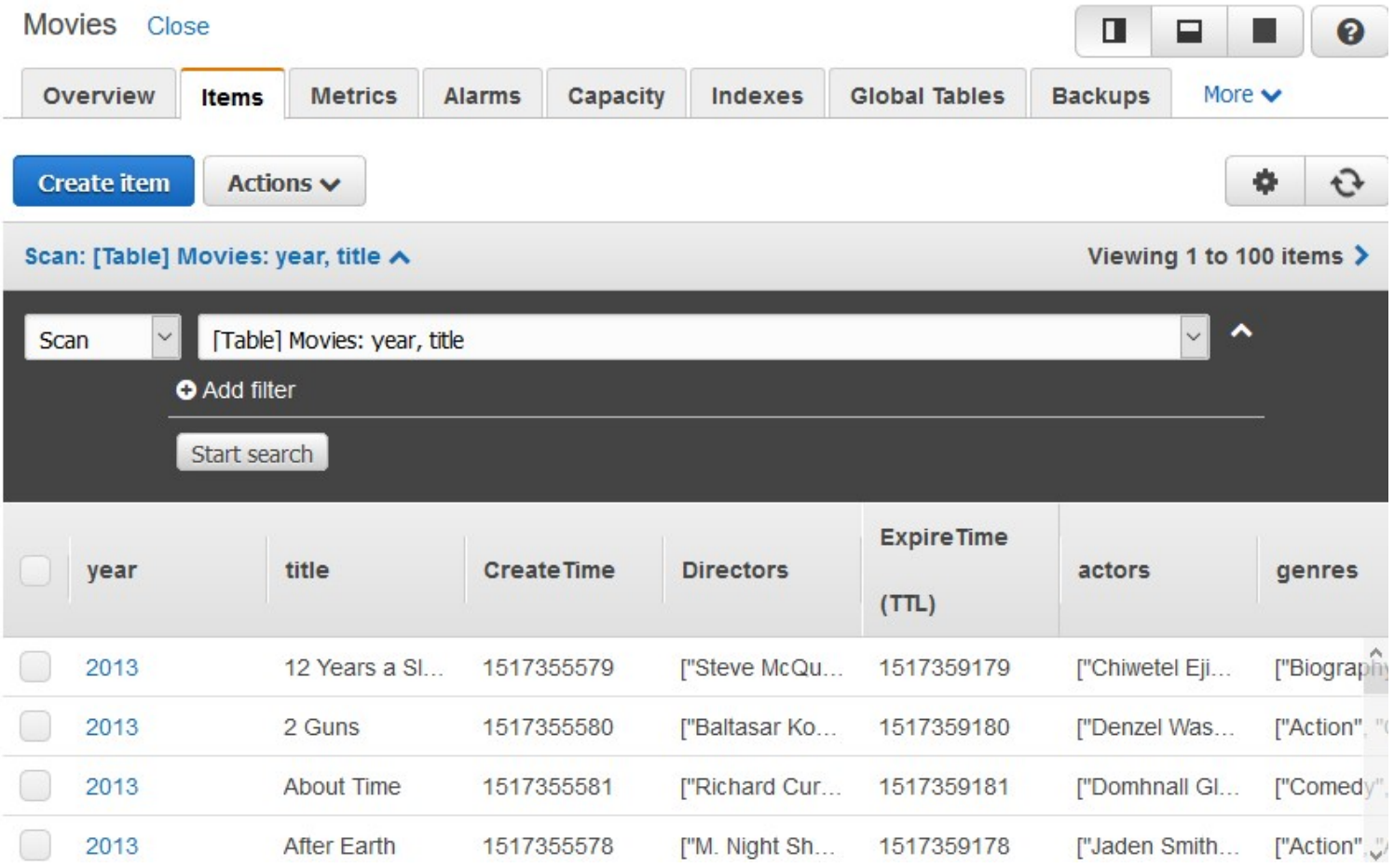
3. Open a command window and execute the following Python script to upload the data.

   ```
   PATH\LoadMovieData.py PATH\moviedata.json
   ```

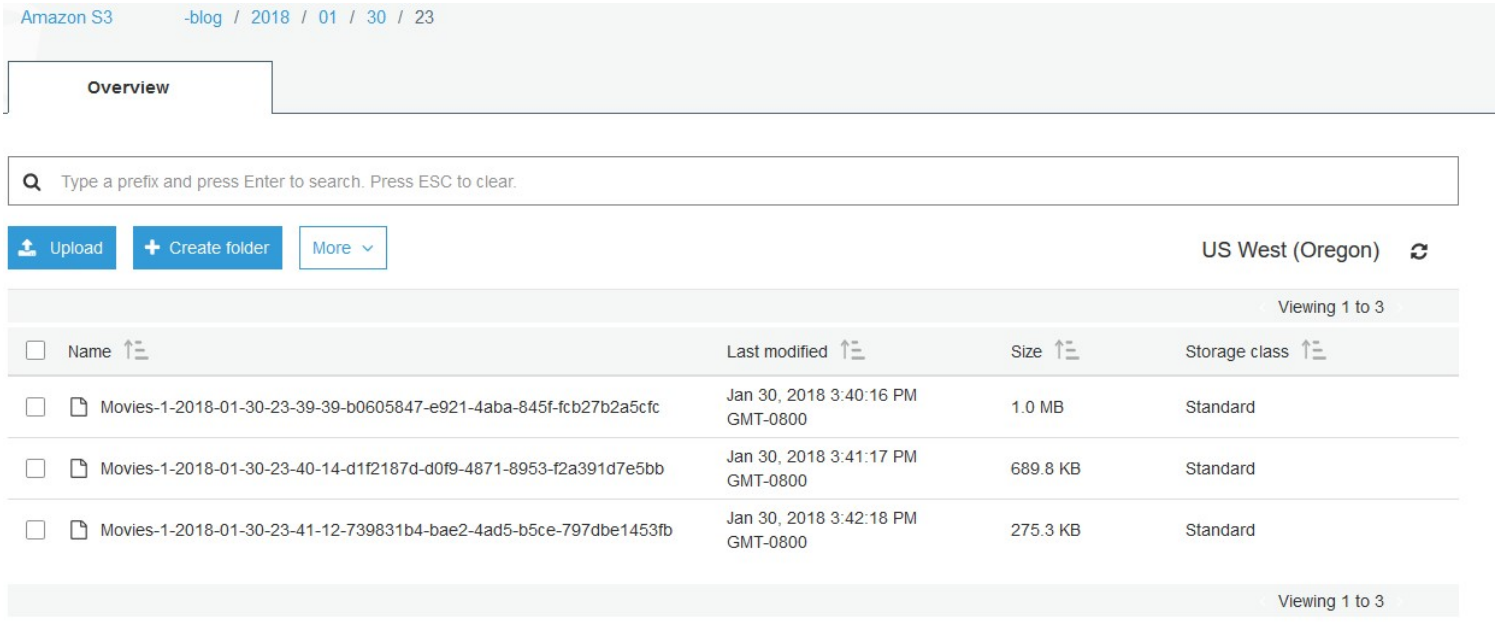
   You should start seeing the following output as records are loaded into the DynamoDB table.

   ```
   10 rows inserted
   20 rows inserted
   30 rows inserted
   …
   4600 rows inserted
   ```

4. Open the DynamoDB console, choose **Tables**, choose the **Movies** table, and choose the **Items**. You should see the first 100 items, as shown in the following screenshot.



5. Open the Amazon S3 console and open the bucket that was created by this solution. You can locate the bucket name by choosing your stack name in the AWS CloudFormation console and choosing the **Resources**. The data in Amazon S3 should appear similar to the following screenshot.



Each object in Amazon S3 contains several movie records based on the Data Firehose buffer settings.

6. Open the Athena console. Choose the database of your choice and paste the following Hive DDL statement, replacing the `LOCATION` with your S3 bucket that contains the movies data. If this is your first time using Athena, see Getting

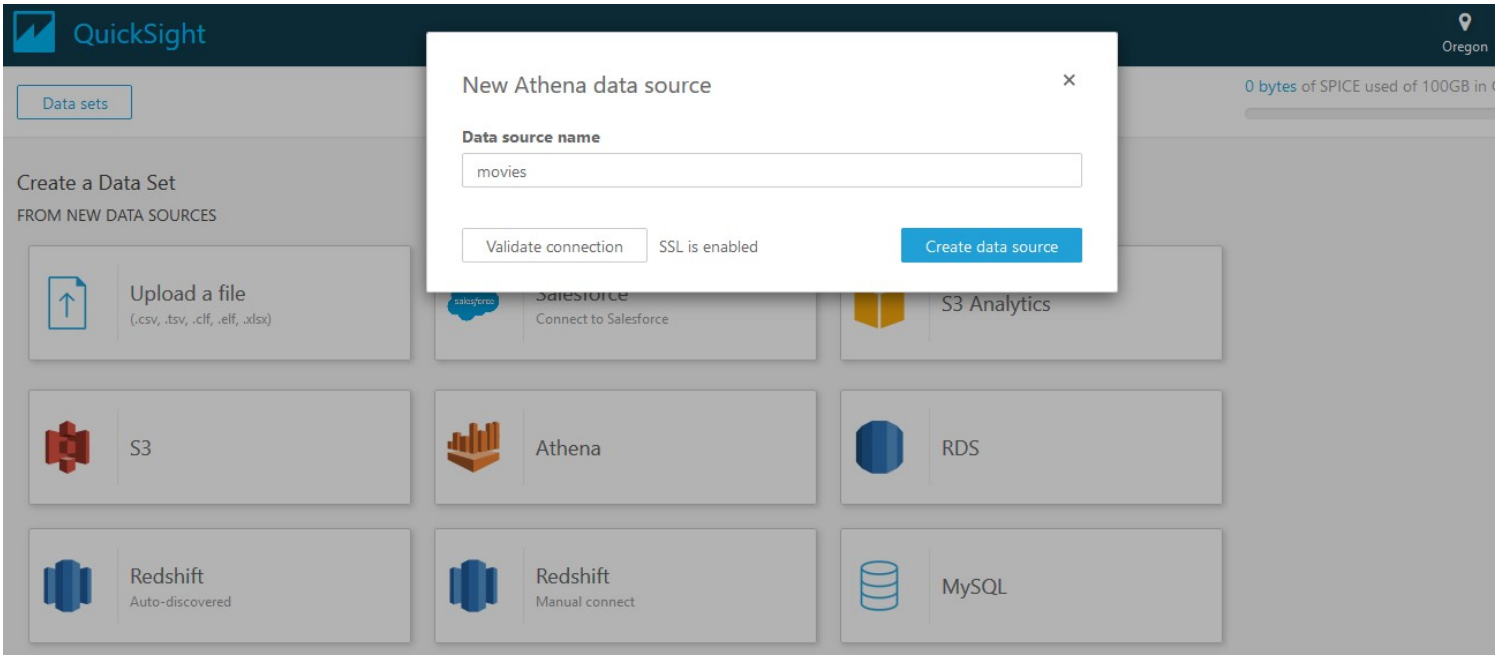
Started in the *Athena User Guide* and create a sample database first.

```
CREATE EXTERNAL TABLE `movies`(
  `year` int,
  `createtime` int,
  `title` string,
  `expiretime` int,
  `info` string,
  `actor1` string,
  `actor2` string,
  `director1` string,
  `director2` string,
  `genre1` string,
  `genre2` string,
  `rating` double)
ROW FORMAT SERDE
  'org.openx.data.jsonserde.JsonSerDe'
LOCATION
  's3://your-bucket-name/path/to/data/'
```

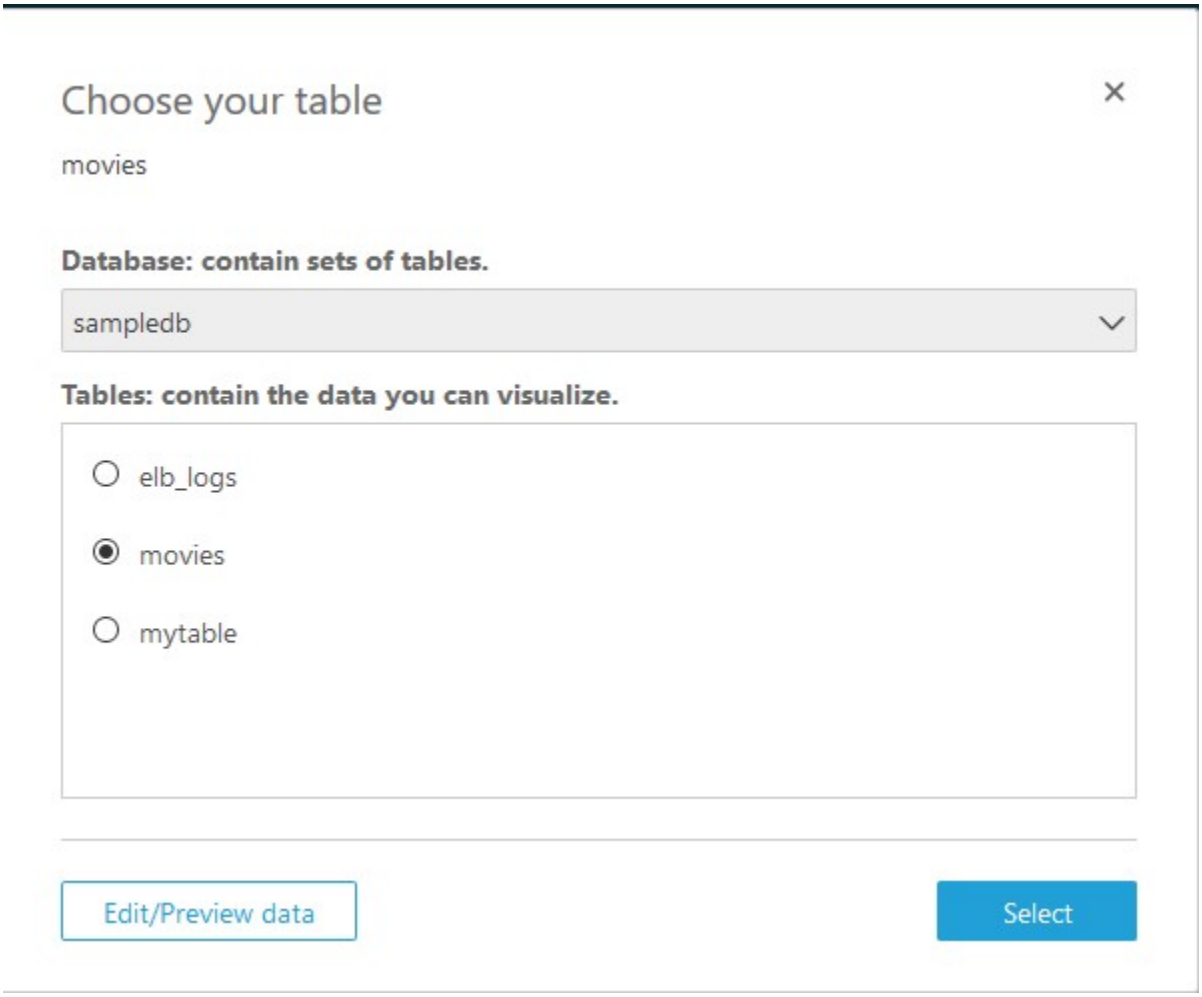
You can query the data from the Athena console. For example, execute the following query to get an average rating by `genre1`.

```
SELECT genre1, avg(rating) as avg_rating FROM "sampledb"."movies" group by genre1 order by avg_
```

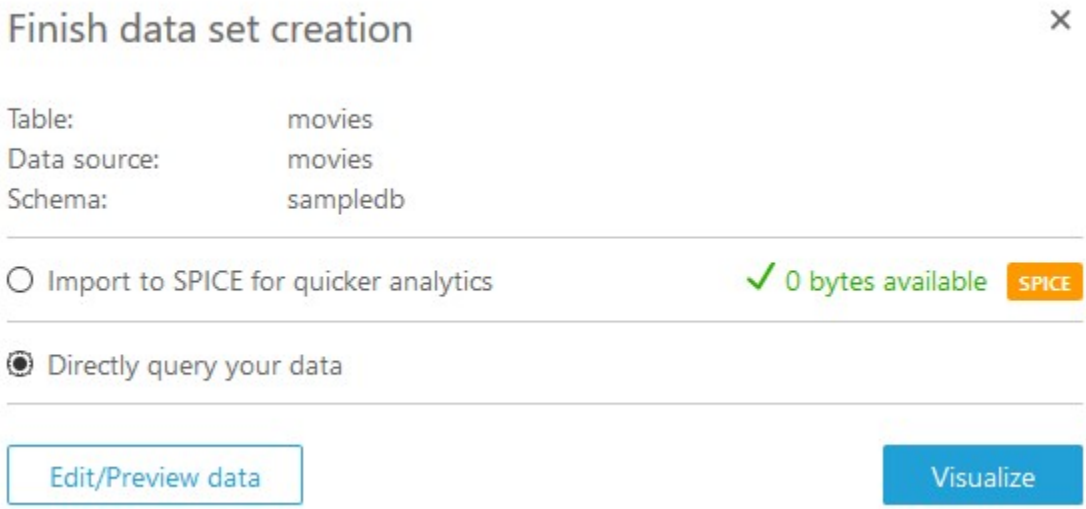
7. Open the Amazon QuickSight console. If this is your first time using Amazon QuickSight, see Getting Started with Data Analysis in Amazon QuickSight in the *Amazon QuickSight User Guide.* Choose **Manage Data** and choose **New data set**. On the **Create a Data Set** page, choose Athena and provide a **Data source name** of your choice (such as `movies`) and choose **Create data source**. You can see an example of this in the following screenshot.



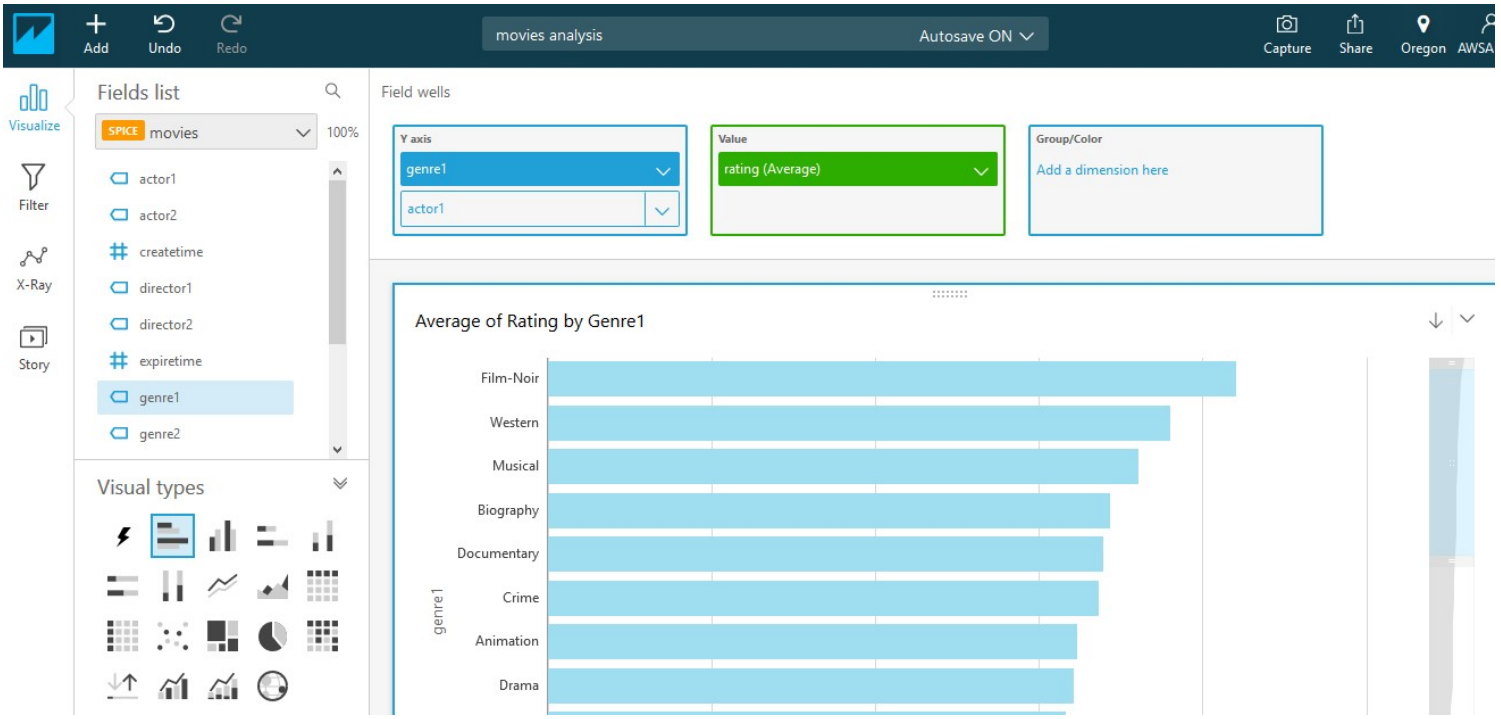
8. On the **Choose your table** page, shown in the image here, choose your database and the **movies** table, and then choose **Select**.



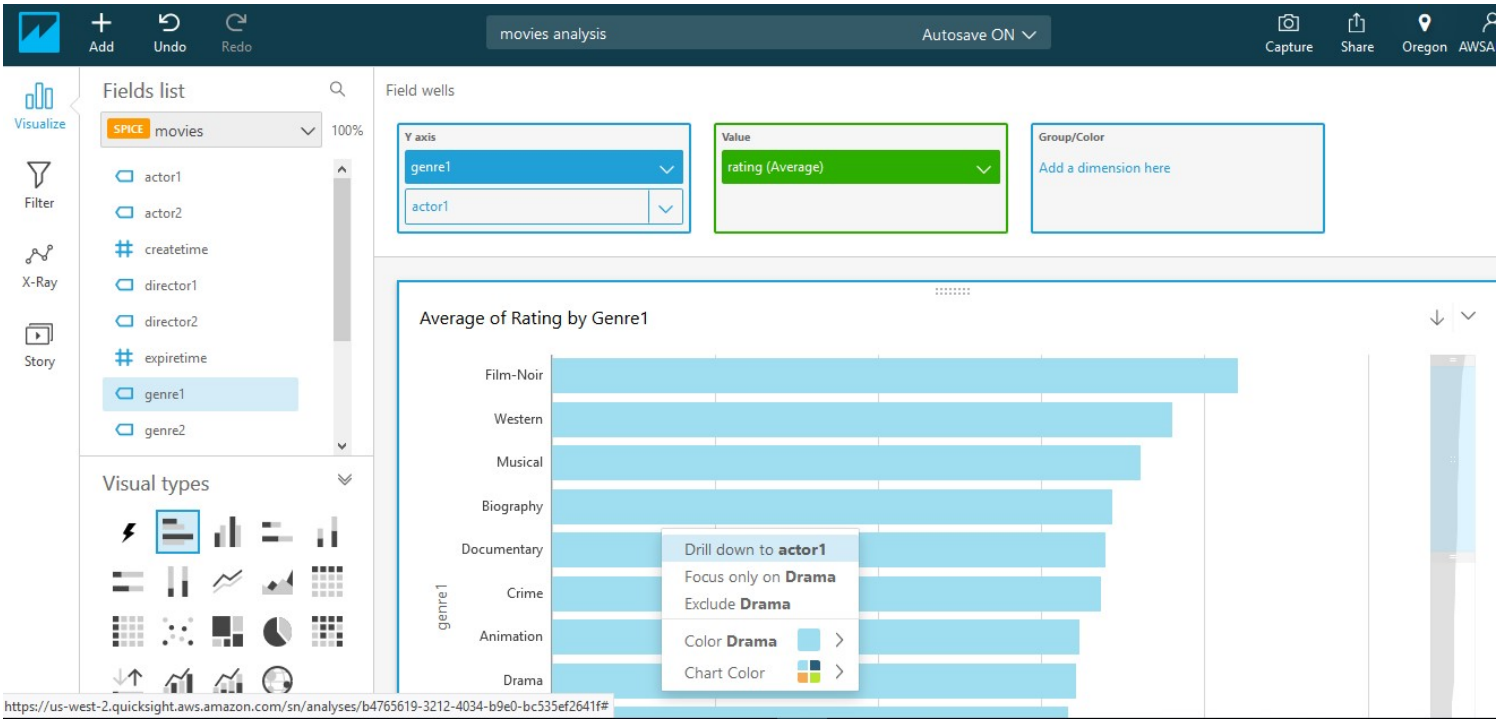
9. On the **Finish data set creation** page, choose **Direct query your data**, and then choose **Visualize**.



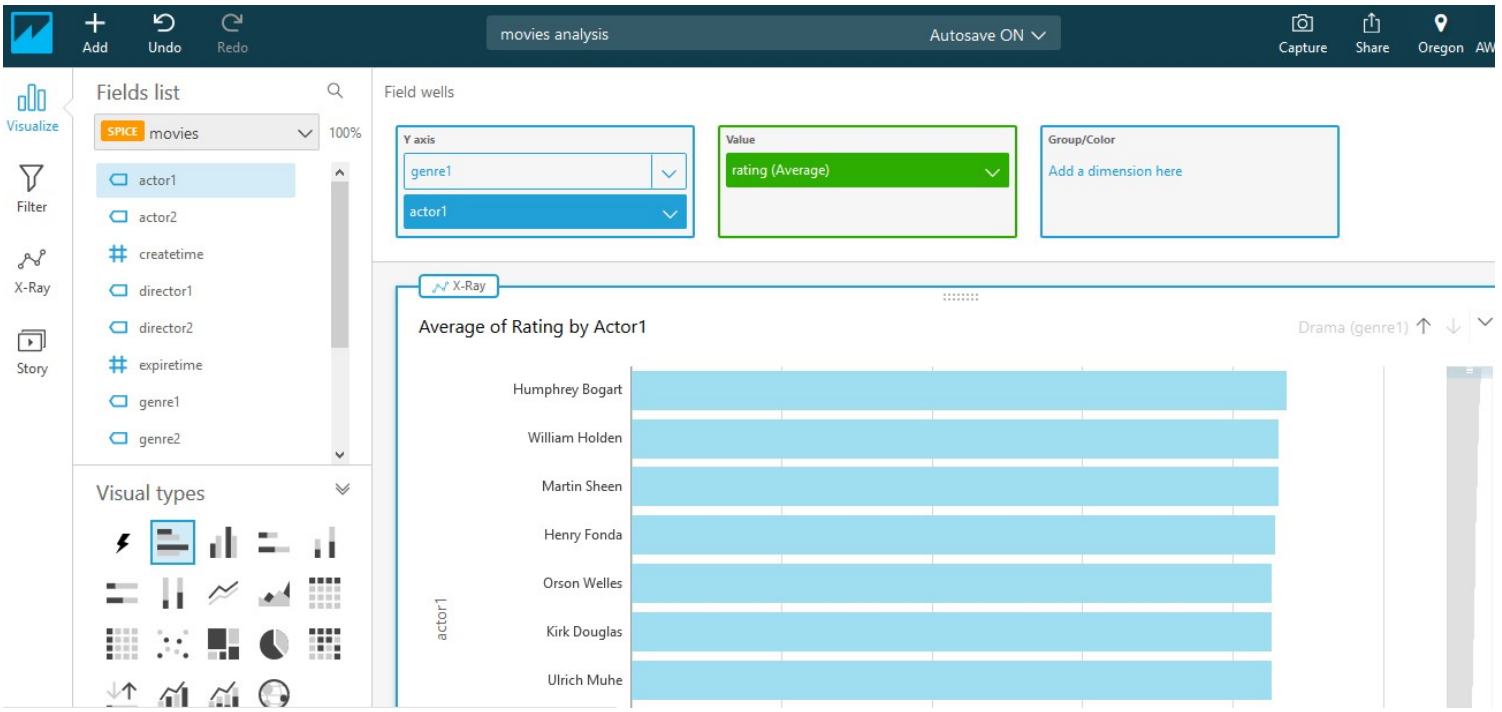
10. You now create a simple visualization. Put `genre1` and `actor1` on the Y-axis and `rating` (average) on the X-axis. The next screenshot shows this.



11. You can now see the average ratings by `genre1`. To view ratings by `actor1`, choose **Drama** and then choose **Drill down to actor1**.



The result shows the average rating by `actor1` view within `genre1`.



Conclusion

The example analytics and visualization solution in this post shows how to tap into the insights contained in your DynamoDB table data. In this post, you build a tiered big data pipeline that enabled you to quickly transition data from your DynamoDB table to an S3 bucket. Then you perform advanced analytics with Athena and a visualization with Amazon QuickSight. This solution has the flexibility to copy all of your DynamoDB data to Amazon S3. You can also move only expired records by using DynamoDB TTL. This type of decoupled solution allows you to make use of the optimal analytics service for each step of the pipeline based on key considerations—latency, throughput, and cost.

About the Authors

Roger Dahlstrom is a solutions architect at Amazon Web Services. He works with the AWS customers to provide guidance and technical assistance on database projects, helping them improve the value of their solutions when using AWS.

Ilya Epshteyn is a principal solutions architect at Amazon Web Services.

TAGS: Amazon Athena, Amazon DynamoDB, Analytics, DynamoDB, visualizations

AWS Training & Certifications

Free digital courses to help you develop your skills

Learn more »

Resources

Getting Started
What's New
Top Posts
Official AWS Podcast
AWS Case Studies

Follow

Twitter
Facebook
LinkedIn
Twitch
RSS Feed
Email Updates

AWS Events

Discover the latest AWS events in your region

Learn more »

Related Posts

Running SQL on Amazon Athena to Analyze Big Data Quickly and Across Regions

New – Announcing Amazon AppFlow

Providing Remote Learning for Millions of Israel Students Through Live Stream

Serving Billions of Ads in Just 100 ms Using Amazon Elasticache for Redis

Query, visualize, and forecast TruFactor web session intelligence with AWS Data Exchange

Using VPC Flow Logs to capture and query EKS network communications

Build a cloud-native network performance analytics solution on AWS for wireless service providers

A public data lake for analysis of COVID-19 data