Amazon Web Services
Athena Workshop

Lab3.Exploring Data Lake with Athena (Advanced)

August 2020

Table of Contents

Lab3.Exploring Data Lake with Athena (Advanced)

Introduction

This additional lab on Amazon Athena covers best practices on partitioning, columnar formats, and compression that can improve query performance and can get significant cost savings.

Prerequisites:

Each student should receive individual AWS Access URL/credentials.

Getting Started

In this lab, you will complete the following tasks: You would use Athena Console to perform all these tasks.

Logon to console and view public dataset in S3:

In this lab, we will be using data from - Amazon Customer Reviews. This dataset provides both TSV (tab separated values) and Parquet versions of over 130 million customer reviews since 1995.

1. To view source dataset in S3, access below URL
   https://console.aws.amazon.com/s3/home?region=us-east-1&bucket=amazon-reviews-pds

   tsv folder has multiple files compressed using gzip. Also notice that file size varies from 12 MB to 2.6 GB.

   Parquet folder has sub-folders on product category and going down one level, you would notice that files are compressed using snappy. File size is more uniform.

Format	Compression	Total Size
Tsv, row storage	gzip	32.2 GB
Parquet, columnar storage	snappy	47.4 GB

   Gzip and Snappy are non-splittable file formats with gzip having higher compression ratio than snappy. But Snappy is significantly faster in compression and decompression speeds.

Compressing your data can speed up your queries significantly, as long as the files are either of an optimal size (see the next section), or the files are splittable. The smaller data sizes reduce network traffic from Amazon S3 to Athena.
Splittable files allow the execution engine in Athena to split the reading of a file by multiple readers to increase parallelism. If you have a single unsplittable file, then only a single reader can read the file while all other readers sit idle. Not all compression algorithms are splittable. The following table lists common compression formats and their attributes.

Algorithm	Splittable?	Compression ratio	Compress + Decompress speed
Gzip (DEFLATE)	No	High	Medium
bzip2	Yes	Very high	Slow
LZO	No	Low	Fast
Snappy	No	Low	Very fast

Queries run more efficiently when reading data can be parallelized and when blocks of data can be read sequentially. Ensuring that your file formats are splittable helps with parallelism regardless of how large your files may be.
However, if your files are too small (generally less than 128 MB), the execution engine might be spending additional time with the overhead of opening Amazon S3 files, listing directories, getting object metadata, setting up data transfer, reading file headers, reading compression dictionaries, and so on. On the other hand, if your file is not splittable and the files are too large, the query processing waits until a single reader has completed reading the entire file. That can reduce parallelism.

Create Athena Tables :

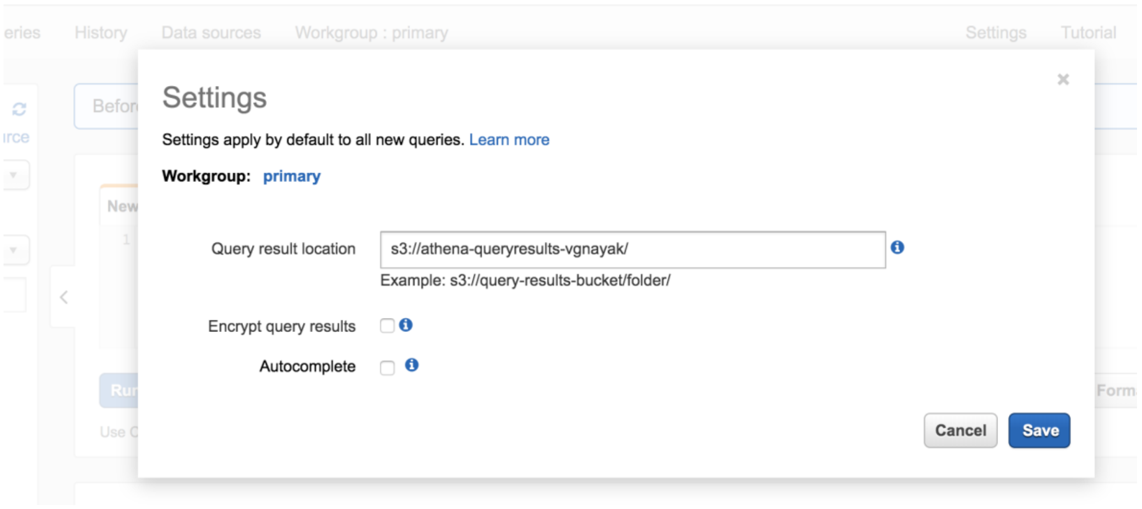
When you create a database and table in Athena, you describe the schema and location of the data, making the data in the table ready for querying.
In this exercise, we will create table using Athena Query editor and then explore an alternate option of automatically creating tables using Glue Crawler.

Once the table is created, it is made available in the centralized Glue Catalog.
Glue catalog is shared by services like Athena, Redshift Spectrum, EMR, Glue ETL and Hive compatible stores.

Create Athena Tables Using Query Editor :

1. Before starting Athena, create s3 bucket to store Athena results. Name : *athena-queryresults-<yourname>*

2. In AWS services, search & select **Athena**.
3. When you get to query editor, go to settings and update Query result location with bucket name created in Step 1.


Settings

Settings apply by default to all new queries. Learn more

Workgroup: primary

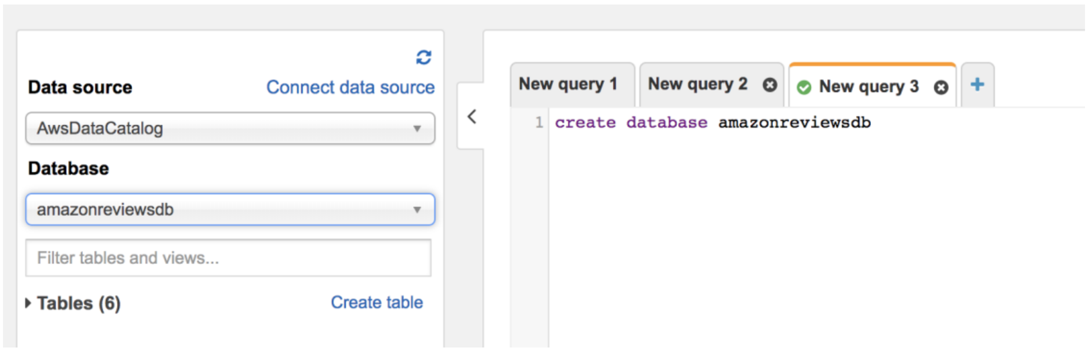
Query result location: s3://athena-queryresults-vgnayak/

Example: s3://query-results-bucket/folder/

Encrypt query results ☐

Autocomplete ☐

4. In query editor, enter *create database amazonreviewsdb*
5. When you run above command, it should create new database.
6. Select your database – *amazonreviewsdb* from Database drop down on the left


Data source: AwsDataCatalog

Database: amazonreviewsdb

```
create database amazonreviewsdb
```

When you run a `CREATE TABLE` query in Athena, you register your table with the AWS Glue Data Catalog. Let's start creating tables.

7. We will create our first table by running following statement that point to TSV files. In create statement LOCATION clause, you'll notice that we are using TSV files.

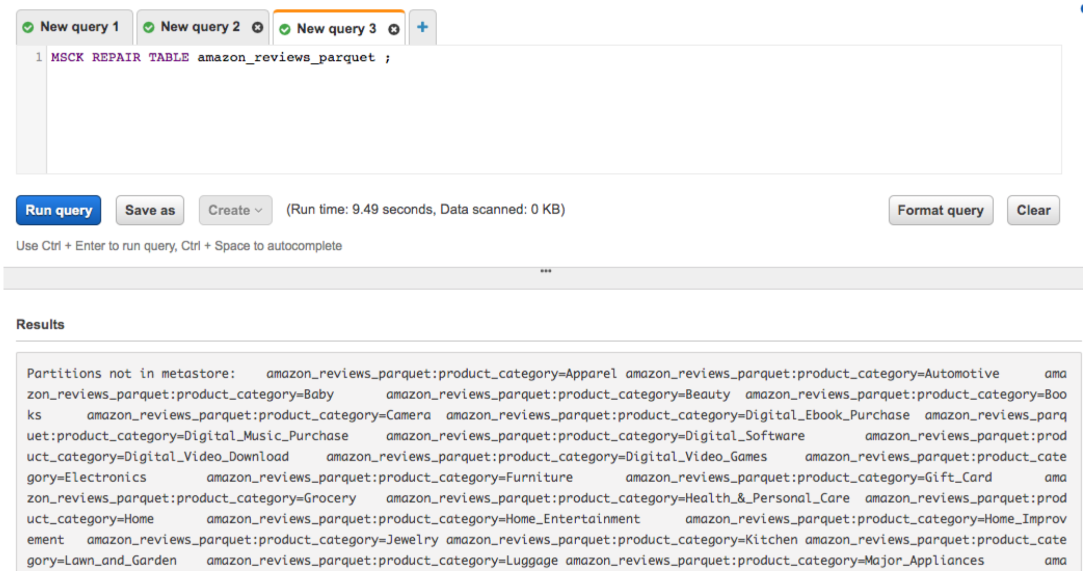
```
CREATE EXTERNAL TABLE amazon_reviews_tsv (
  marketplace string,
  customer_id string,
  review_id string,
  product_id string,
  product_parent string,
  product_title string,
  product_category string,
  star_rating int,
  helpful_votes int,
  total_votes int,
  vine string,
  verified_purchase string,
  review_headline string,
  review_body string,
  review_date date,
  year int)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
  ESCAPED BY '\\'
  LINES TERMINATED BY '\n'
LOCATION
  's3://amazon-reviews-pds/tsv/'
TBLPROPERTIES ("skip.header.line.count"="1");
```

8. Create the second table in the database, this time pointing to parquet files

```
CREATE EXTERNAL TABLE `amazon_reviews_parquet`(
  `marketplace` string,
  `customer_id` string,
  `review_id` string,
  `product_id` string,
  `product_parent` string,
  `product_title` string,
  `star_rating` int,
  `helpful_votes` int,
  `total_votes` int,
  `vine` string,
  `verified_purchase` string,
  `review_headline` string,
  `review_body` string,
  `review_date` bigint,
  `year` int)
PARTITIONED BY ( `product_category` string )
STORED AS PARQUET
LOCATION 's3://amazon-reviews-pds/parquet';
```

9. In the previous step, although query is successful you could notice a message stating that partitions need to be loaded to this table. To load the partitions, run below statement

```
MSCK REPAIR TABLE amazon_reviews_parquet ;
```



```
MSCK REPAIR TABLE amazon_reviews_parquet ;
```

(Run time: 9.49 seconds, Data scanned: 0 KB)

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

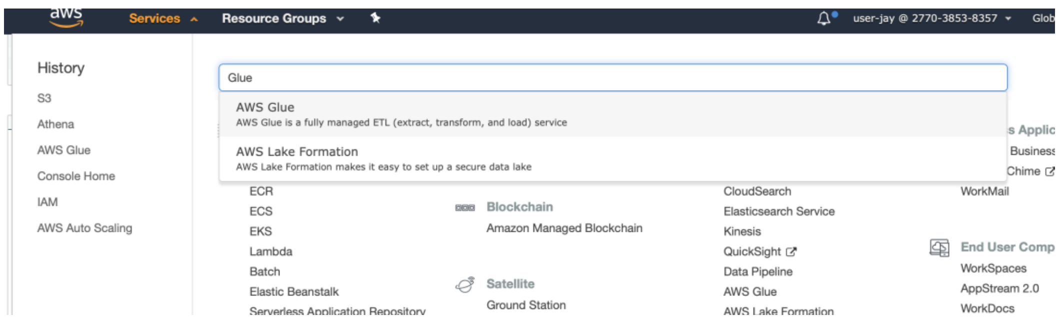
```
Partitions not in metastore:    amazon_reviews_parquet:product_category=Apparel amazon_reviews_parquet:product_category=Automotive      ama
zon_reviews_parquet:product_category=Baby        amazon_reviews_parquet:product_category=Beauty  amazon_reviews_parquet:product_category=Boo
ks      amazon_reviews_parquet:product_category=Camera  amazon_reviews_parquet:product_category=Digital_Ebook_Purchase  amazon_reviews_parq
uet:product_category=Digital_Music_Purchase     amazon_reviews_parquet:product_category=Digital_Software        amazon_reviews_parquet:prod
uct_category=Digital_Video_Download     amazon_reviews_parquet:product_category=Digital_Video_Games     amazon_reviews_parquet:product_cate
gory=Electronics        amazon_reviews_parquet:product_category=Furniture       amazon_reviews_parquet:product_category=Gift_Card       ama
zon_reviews_parquet:product_category=Grocery    amazon_reviews_parquet:product_category=Health_&_Personal_Care  amazon_reviews_parquet:prod
uct_category=Home       amazon_reviews_parquet:product_category=Home_Entertainment      amazon_reviews_parquet:product_category=Home_Improv
ement   amazon_reviews_parquet:product_category=Jewelry amazon_reviews_parquet:product_category=Kitchen amazon_reviews_parquet:product_cate
gory=Lawn_and_Garden    amazon_reviews_parquet:product_category=Luggage amazon_reviews_parquet:product_category=Major_Appliances        ama
```

Athena leverages Hive for partitioning data. You can partition your data by any key. Also, if you query a partitioned table and specify the partition in the WHERE clause, Athena scans the data only from that partition thus leading to less cost per query.

Create Athena Tables Using Glue Crawler :

Now, let's leverage crawler in AWS Glue to retrieve schema information automatically. The crawlers go through your data, and inspect portions of it to determine the schema. In addition, the crawler can detect and register partitions.

1. In the AWS services console, search for Glue and select AWS Glue

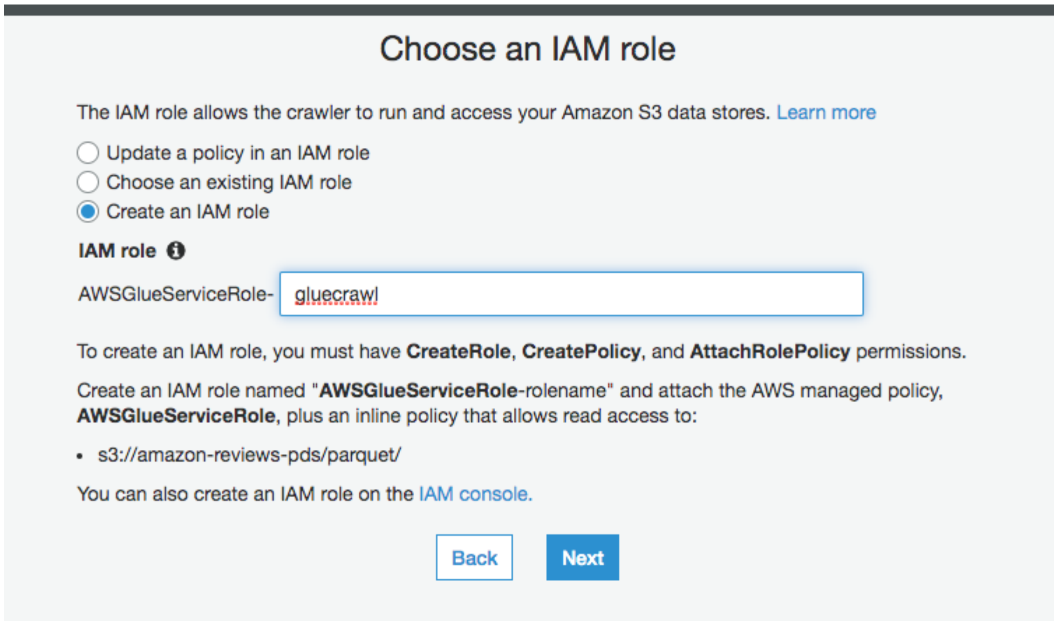


AWS Glue
AWS Glue is a fully managed ETL (extract, transform, and load) service

AWS Lake Formation
AWS Lake Formation makes it easy to set up a secure data lake

2. In the left pane, click "crawlers" and then select "Add Crawler".
3. Enter the crawler name "*amazon_reviews_parquet_gluecrawler*" and select Next.
4. Select "Data Stores" and select Next.
5. Enter the S3 location as "*s3://amazon-reviews-pds/parquet/*" and select Next

6. Select Next on "Add Another data store" with default option selected as No
7. Now create a new IAM role for the crawler and hit Next



Choose an IAM role

The IAM role allows the crawler to run and access your Amazon S3 data stores. Learn more

- Update a policy in an IAM role
- Choose an existing IAM role
- [✓ Create an IAM role]

IAM role

AWSGlueServiceRole- gluecrawl

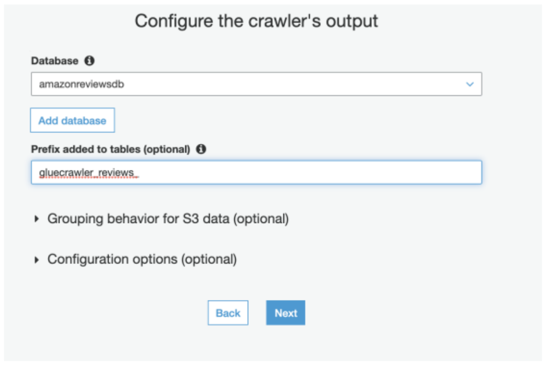
To create an IAM role, you must have **CreateRole**, **CreatePolicy**, and **AttachRolePolicy** permissions.

Create an IAM role named "**AWSGlueServiceRole**-rolename" and attach the AWS managed policy, **AWSGlueServiceRole**, plus an inline policy that allows read access to:

- s3://amazon-reviews-pds/parquet/

You can also create an IAM role on the IAM console.

8. Choose a schedule for crawler. For this exercise just select "Run on Demand" and hit next.
9. Select database "*amazonreviewsdb*", enter table name prefix "*gluecrawler_reviews_*" . Click finish to Add the crawler.



Configure the crawler's output

Database

amazonreviewsdb

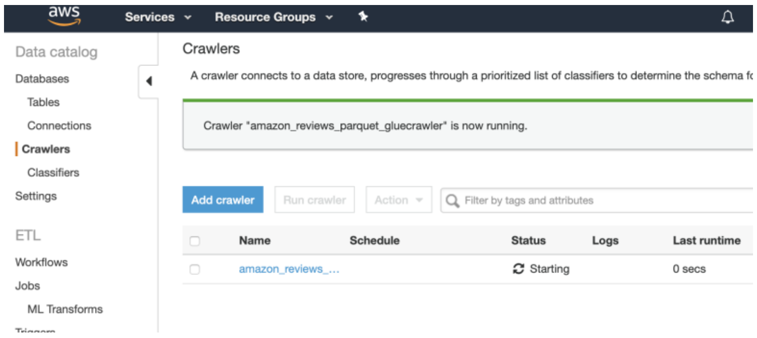
Prefix added to tables (optional)

gluecrawler_reviews_

Add database
Grouping behavior for S3 data (optional)
Configuration options (optional)
Back
Next

10. Find the new crawler listed with status "Ready". Select the checkbox next to the crawler and select "RunCrawler". The crawler would show initial "**starting**" status and it would turn to "stopping" and then turn to "**Ready**". This should create table in catalog. *Approximate run time ~ 3 mins.*

Lab3.Exploring Data Lake with Athena (Advanced)

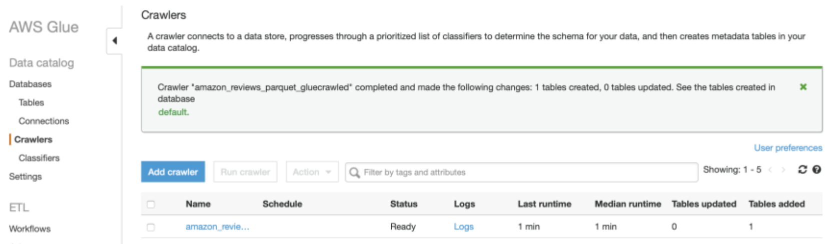


Crawlers

A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema fo

Crawler "amazon_reviews_parquet_gluecrawler" is now running.

	Name	Schedule	Status	Logs	Last runtime
	amazon_reviews_...		Starting		0 secs



Crawlers

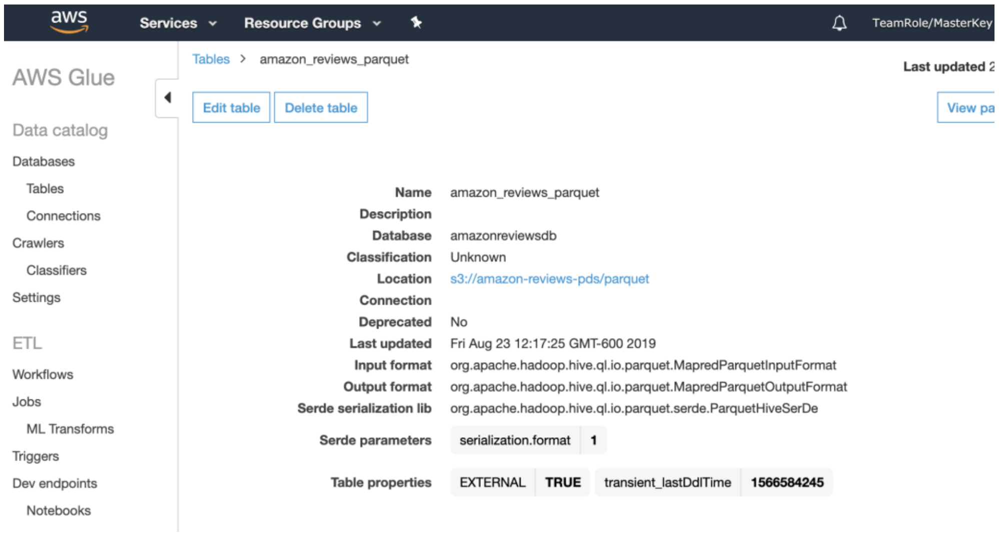
A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.

Crawler "amazon_reviews_parquet_gluecrawled" completed and made the following changes: 1 tables created, 0 tables updated. See the tables created in database default.

	Name	Schedule	Status	Logs	Last runtime	Median runtime	Tables updated	Tables added
	amazon_revie...		Ready	Logs	1 min	1 min	0	1

Verify Tables in Glue Catalog

1. If you are in AWS Glue, then navigate to tables and you should see table created by crawler with prefix - gluecrawler_reviews_
2. Also, you should see tables created in earlier step from Athena query editor
3. Select table "amazon_reviews_parquet" created in first step and view the table definitions. Notice the location where the table data is referred.

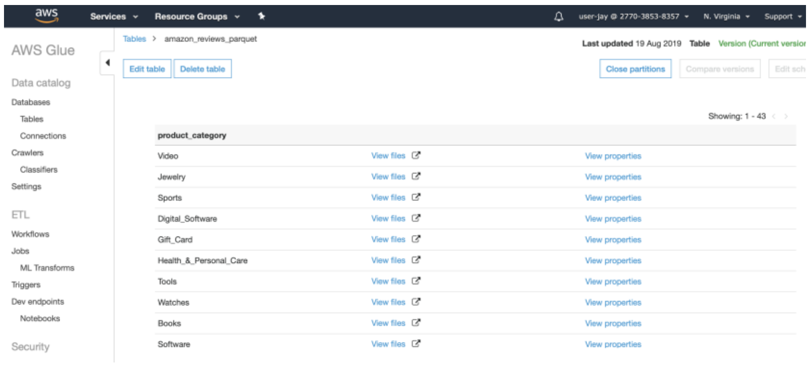


Tables > amazon_reviews_parquet

Last updated 2

Name	amazon_reviews_parquet
Description	
Database	amazonreviewsdb
Classification	Unknown
Location	s3://amazon-reviews-pds/parquet
Connection	
Deprecated	No
Last updated	Fri Aug 23 12:17:25 GMT-600 2019
Input format	org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat
Output format	org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat
Serde serialization lib	org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe
Serde parameters	serialization.format 1
Table properties	EXTERNAL TRUE transient_lastDdlTime 1566584245

At the right top, click on "view partitions ", to view the various partition values available for product_category.



Last updated 19 Aug 2019 **Table** Version (Current versior

Showing: 1 - 43

product_category		
Video	View files	View properties
Jewelry	View files	View properties
Sports	View files	View properties
Digital_Software	View files	View properties
Gift_Card	View files	View properties
Health_&_Personal_Care	View files	View properties
Tools	View files	View properties
Watches	View files	View properties
Books	View files	View properties
Software	View files	View properties

Query Data with Amazon Athena

Let's start running queries

1. In Athena Query Editor, select the database "amazonreviewsdb"
2. Now let's run a simple query on tsv table to preview data

```
SELECT * FROM "amazon_reviews_tsv"
WHERE marketplace = 'US'
limit 10;
```

Run time: 2.66 seconds, Data scanned: 236.87 KB

3. Now let's try a query with aggregations on tsv and parquet tables. This is common pattern in analytical/reporting applications.
First, let's run this on tsv to capture run time and data scanned

```
SELECT product_id, product_title, count(*) as num_reviews, avg(star_rating) as
avg_stars
FROM amazon_reviews_tsv where product_category='Toys'
GROUP BY 1, 2
ORDER BY 3 DESC
limit 100;
```

(Run time: 1 minute 26 seconds, Data scanned: 32.23 GB)

4. Run the same query now against Parquet partitioned table

```
SELECT product_id, product_title, count(*) as num_reviews, avg(star_rating) as
avg_stars
FROM amazon_reviews_parquet where product_category='Toys'
GROUP BY 1, 2
ORDER BY 3 DESC
limit 100;
```

(Run time: 4.53 seconds, Data scanned: 215.04 MB)

Significant improvement in performance and data scanned can be attributed to parquet columnar storage format and effective partitions on predicates. When Athena executes a query on a partitioned table, it first checks to see if any partitioned columns were used in the WHERE clause of the query. If partitioned columns were used, Athena requests the AWS Glue Data Catalog to return the partition specification matching the specified partition columns. The partition specification includes the LOCATION property that tells Athena which Amazon S3 prefix to use when reading data. In this case, *only* data stored in this prefix is scanned.

Create view

1. A view in Amazon Athena is a logical, not a physical table. The query that defines a view runs each time the view is referenced in a query. Create views when you want to hide underlying complexity and minimize maintenance problems if underlying table/column names change.
2. Open a new Athena Query editor tab and run the below query to retrieve TopReviewedStarRated products

```
Create view TopReviewedStarRatedProducts_v as
SELECT product_category, product_id, product_title, count(*) TotalReviews  FROM
amazon_reviews_parquet
WHERE  star_rating=5
group by product_category, product_id, product_title
order by TotalReviews desc
```

3. You can now run query on this view to extract anytime the Top N reviewed products among those high rated over the years.

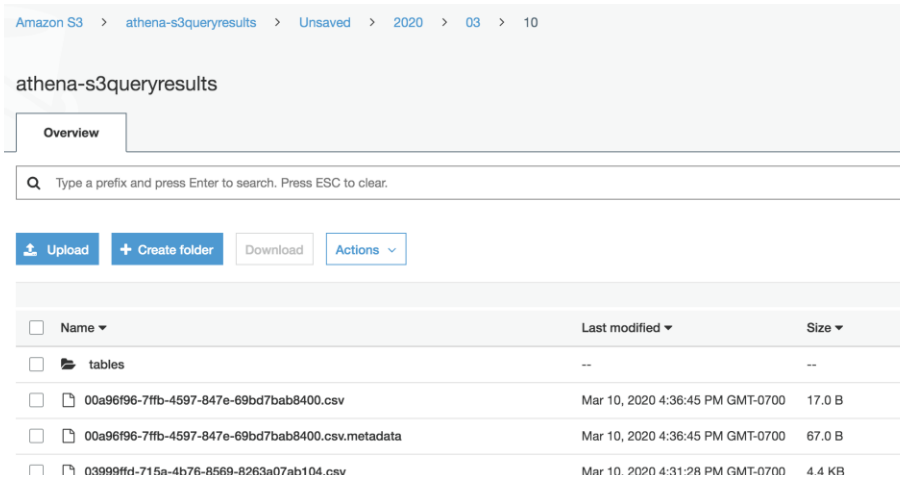
```
select * from topreviewedstarratedproducts_v
 limit 10
```

Run time: 6.06 seconds, Data scanned: 6.39 GB

Query Results and History

1. For every query run, Athena automatically saves query results in S3 at query result location. Files are saved to the query result location in Amazon S3 based on the name of the query, the query ID, and the date that the query ran. Files for each query are named using the QueryID, which is a unique identifier that Athena assigns to each query when it runs.
2. To identify QueryID – Choose history from Query Editor and then click on state – “Failed” or “Succeeded”

3. Navigate to S3 location *athena-queryresults-<yourname>* and identify the query results saved in S3 for last query execution.

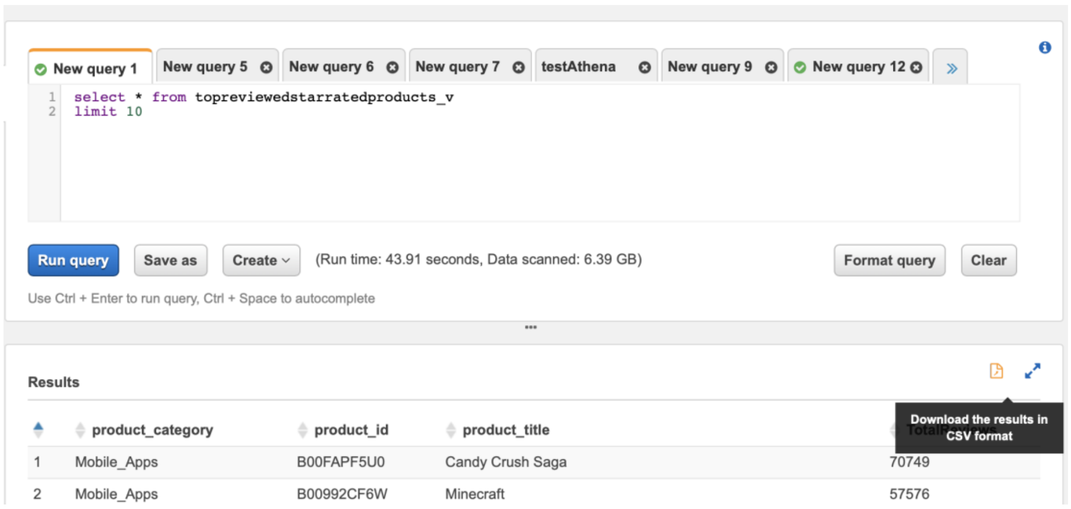

Amazon S3 > athena-s3queryresults > Unsaved > 2020 > 03 > 10

athena-s3queryresults

Overview
Type a prefix and press Enter to search. Press ESC to clear.
Upload
Create folder
Download
Actions

Name	Last modified	Size
tables	--	--
00a96f96-7ffb-4597-847e-69bd7bab8400.csv	Mar 10, 2020 4:36:45 PM GMT-0700	17.0 B
00a96f96-7ffb-4597-847e-69bd7bab8400.csv.metadata	Mar 10, 2020 4:36:45 PM GMT-0700	67.0 B
03999ffd-715a-4b76-8569-8263a07ab104.csv	Mar 10, 2020 4:31:28 PM GMT-0700	4.4 KB

4. You can also optionally download the results from Query editor via the download link



```
select * from topreviewedstarratedproducts_v
limit 10
```

(Run time: 43.91 seconds, Data scanned: 6.39 GB)

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

Results

	product_category	product_id	product_title	Total
1	Mobile_Apps	B00FAPF5U0	Candy Crush Saga	70749
2	Mobile_Apps	B00992CF6W	Minecraft	57576

Run CTAS to create new partitioned parquet table

In earlier step, we started by accessing parquet formatted dataset directly in Athena. But there will be instances where files are staged in CSV/TSV/TXT and we need to run a step of converting them to parquet/orc format.

Objective of this section is to show you an approach to accomplish this conversion.

We would use CTAS (Create Table As Select) command to create a new table that first queries TSV data and writes the results in Parquet format in S3. Also partitions the dataset by "marketplace" and "year"

Lab3.Exploring Data Lake with Athena (Advanced)

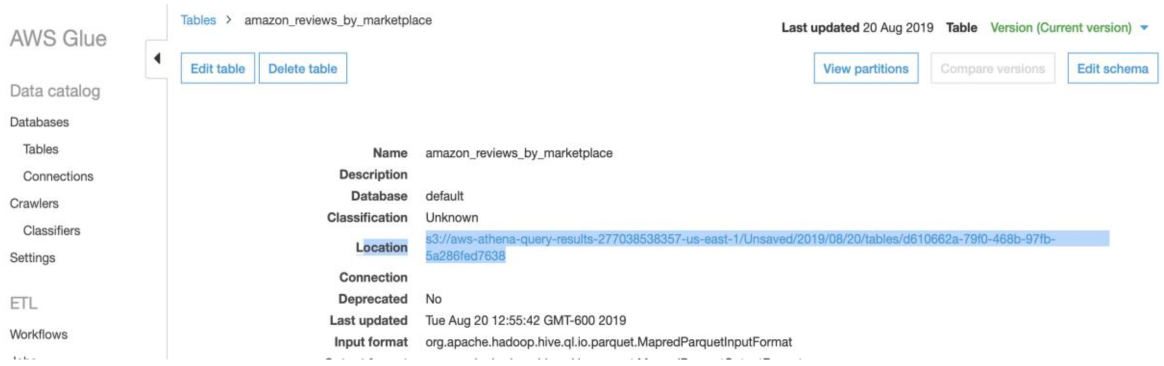
1. Run below query in Athena. This query takes ~5 mins to complete.

```
CREATE TABLE amazon_reviews_by_marketplace
WITH (
  format='PARQUET',
  partitioned_by = ARRAY['marketplace', 'year']
) AS
SELECT customer_id, review_id, product_id, product_parent, product_title,
product_category, star_rating, helpful_votes, total_votes, verified_purchase,
review_headline, review_body, review_date,
  marketplace,
  year(review_date) as year
FROM amazon_reviews_tsv
WHERE "$path" LIKE '%tsv.gz'
```

Note: CTAS command writes results to S3 under S3://*athena-queryresults-<yourname>/* location. You can specify your own S3 location by specifying
   external_location = 's3://... ', in the CTAS command.

Partitioning strategy depends on the query access patterns. In this case, we want to analyze the data by marketplace and year. The following queries will show how proper partitioning can help reduce data scan and improve performance for this access pattern.

2. Once table is created, go to glue catalog and view the newly created table. Select the new table "amazon_reviews_by_marketplace".
3. Find the S3 location where the table has created new partitioned data.



Last updated 20 Aug 2019 Table Version (Current version)

Name	amazon_reviews_by_marketplace
Description	
Database	default
Classification	Unknown
Location	s3://aws-athena-query-results-277038538357-us-east-1/Unsaved/2019/08/20/tables/d610662a-79f0-468b-97fb-5a286fed7638
Connection	
Deprecated	No
Last updated	Tue Aug 20 12:55:42 GMT-600 2019
Input format	org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat

4. Click the S3 location to view the new partitioned structure. The first level partition displays "marketplace" and selecting one of the folders display its inner partition by "year" and the contents within.

5. Now go back to Athena Query Editor and try the following queries over the newly created table.

```
SELECT product_id, COUNT(*) FROM amazon_reviews_by_marketplace
WHERE marketplace='US' AND year = 2013
GROUP BY 1 ORDER BY 2 DESC LIMIT 10
```

[parquet + partitioned by marketplace] - > (Run time: 5.27 seconds, Data scanned: 145.66 MB)

Vs

```
SELECT product_id, COUNT(*) FROM amazon_reviews_parquet
WHERE marketplace='US' AND year = 2013
GROUP BY 1 ORDER BY 2 DESC LIMIT 10
```

[parquet] - > (Run time: 5.88 seconds, Data scanned: 882 MB). Though performance is comparable, data scan is high

vs

```
SELECT product_id, COUNT(*) FROM amazon_reviews_tsv
WHERE marketplace='US' AND extract(year from review_date) = 2013
GROUP BY 1 ORDER BY 2 DESC LIMIT 10
```

[tsv] -> (Run time: 1 minute 43 seconds, Data scanned: 32.23 GB). Performance poor and data scan is higher.

In above examples, we saw partitioning saves you query costs and query time.
But how would you optimize if you have high cardinality fields? (large number of distinct values). For such need, we can explore bucketing. With this data can be split for storage into many buckets that will have roughly the same amount of data. Please note that Athena leverages Hive bucketing, which is different from S3 buckets.

Run CTAS to explore bucketing

1. Let's assume a case where we have need to query by Review_id more often to pull its specific information. Since Review_id has high cardinality, bucketing is better strategy to split the data. But before optimizing, let's run the below query on original TSV data to fetch product ID by review Id.

```
SELECT  "$path", product_id FROM amazon_reviews_tsv
WHERE review_id='RWKE7RNMWTQYT'
```

Query Takes Run time: 1 minute 30 seconds, Data scanned: 32.21 GB

Note: **"$path"** returns the path in S3 where the data for the review_id is located.

2. Now run the same query on amazon_reviews_by_marketplace query which is already partitioned by marketplace and year and is parquet.

```
SELECT  "$path", product_id FROM amazon_reviews_by_marketplace
WHERE review_id='RWKE7RNMWTQYT'
```

This query takes Run time: 5.51 seconds, Data scanned: **1.46 GB**

parquet being columnar format is able to retrieve results much faster.

3. Now let's use CTAS to create bucketed table on "Review_ID" as our query is just by review_id. We can't use normal partition as it would result in creating too many partitions and fail on max limit.

```
CREATE TABLE amazon_reviews_unsorted
WITH (
  format='Parquet',
  bucketed_by = ARRAY['review_id'],
  bucket_count = 30
) AS
SELECT review_id, product_id, customer_id, product_parent, star_rating, helpful_votes,
total_votes, verified_purchase, marketplace, product_category, review_date
FROM amazon_reviews_by_marketplace
```

4. Now try the query on the bucket partitioned table and check it run time stats.

```
SELECT  "$path", product_id FROM amazon_reviews_unsorted
WHERE review_id='RWKE7RNMWTQYT'
```

Run time: 3.43 seconds, Data scanned: 51.92 MB

As we have bucketed data on reviewId, Athena only queries the specific buckets by first checking the parquet meta data thereby reducing time and scan size.

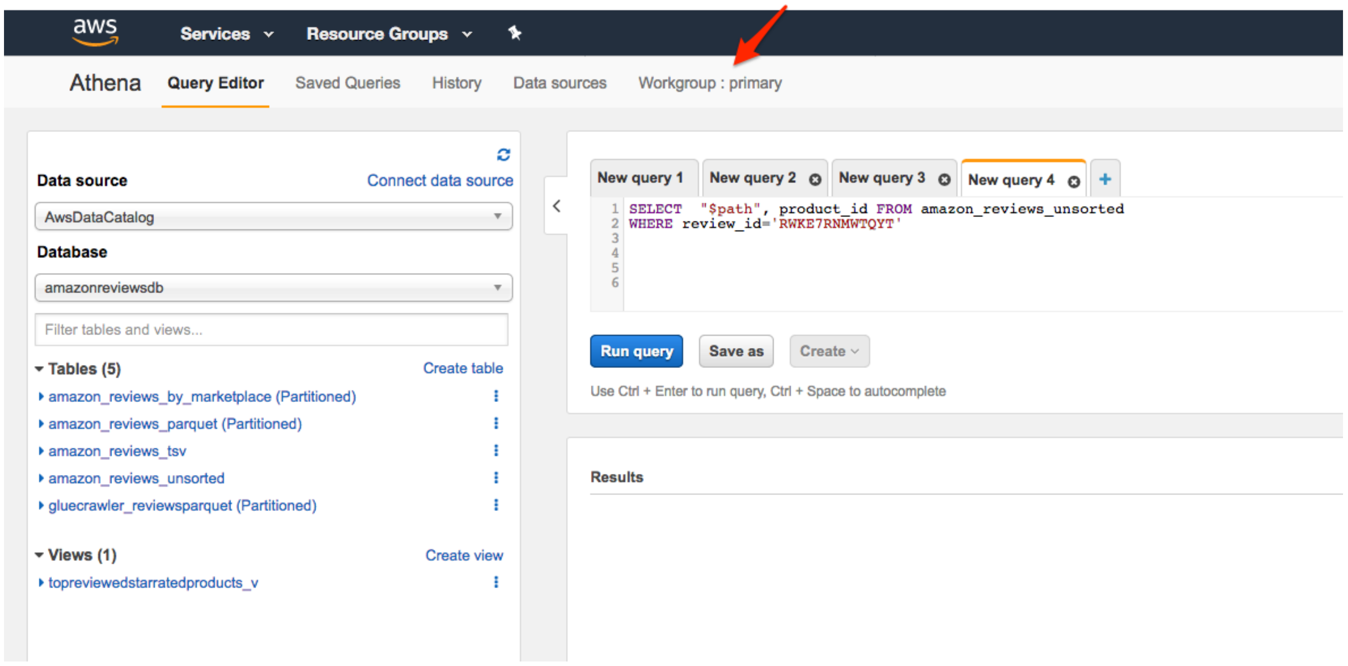
Athena Workgroups

Use workgroups to separate users, teams, applications, or workloads, to set limits on amount of data each query or the entire workgroup can process, and to track costs. Because workgroups act as resources, you can use resource-level identity-based policies to control access to a specific workgroup

Lab3.Exploring Data Lake with Athena (Advanced)

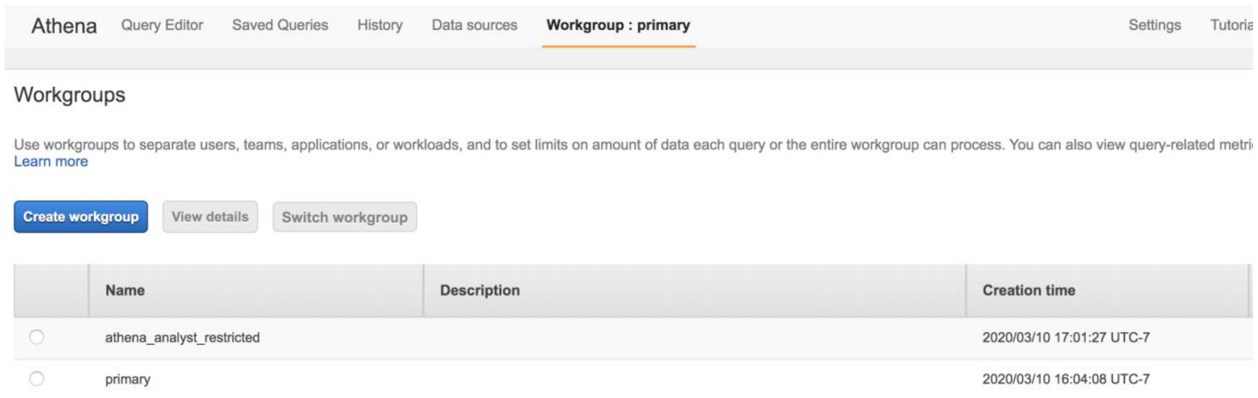
Creating workgroup and setting controls

1. In this lab, we will create a new workgroup and define data usage controls.
2. Go to Athena query editor and click on workgroup: primary



```
SELECT  "$path", product_id FROM amazon_reviews_unsorted
WHERE review_id='RWKE7RNMWTQYT'
```

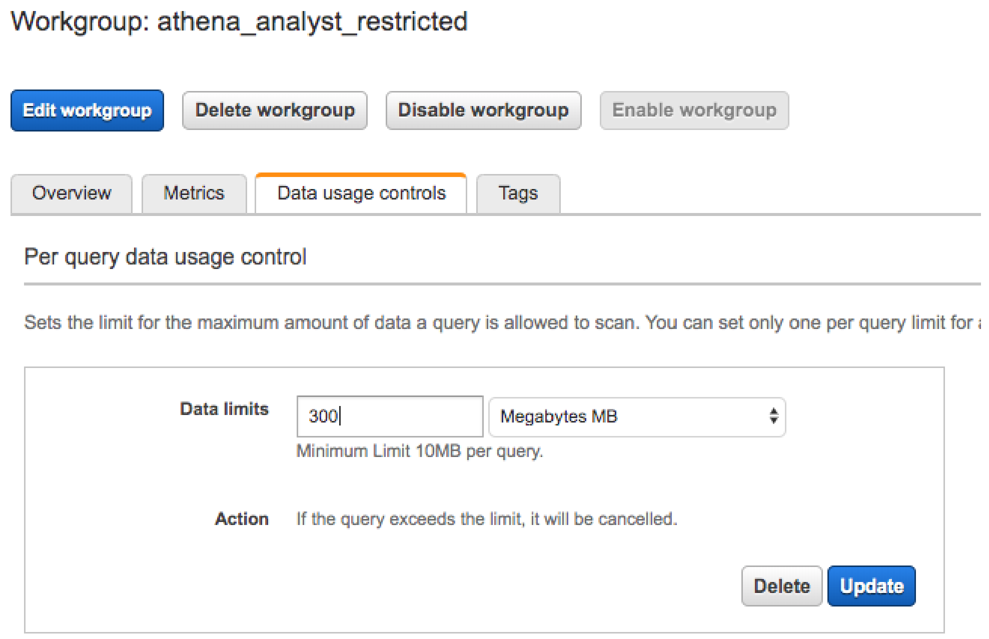
3. Select "create workgroup" and provide workgroup name - "*athena_analyst_restricted*". Add Query result location. Other options can be left default. Go ahead and select "create workgroup".



Use workgroups to separate users, teams, applications, or workloads, and to set limits on amount of data each query or the entire workgroup can process. You can also view query-related metri
Learn more

	Name	Description	Creation time
○	athena_analyst_restricted		2020/03/10 17:01:27 UTC-7
○	primary		2020/03/10 16:04:08 UTC-7

4. Once group is created. Select the group and view details. Select tab "data usage controls" . Enter 300 in Data limits for Megabytes MB and update.



Workgroup: athena_analyst_restricted

Per query data usage control

Sets the limit for the maximum amount of data a query is allowed to scan. You can set only one per query limit for

Data limits: 300 Megabytes MB

Minimum Limit 10MB per query.

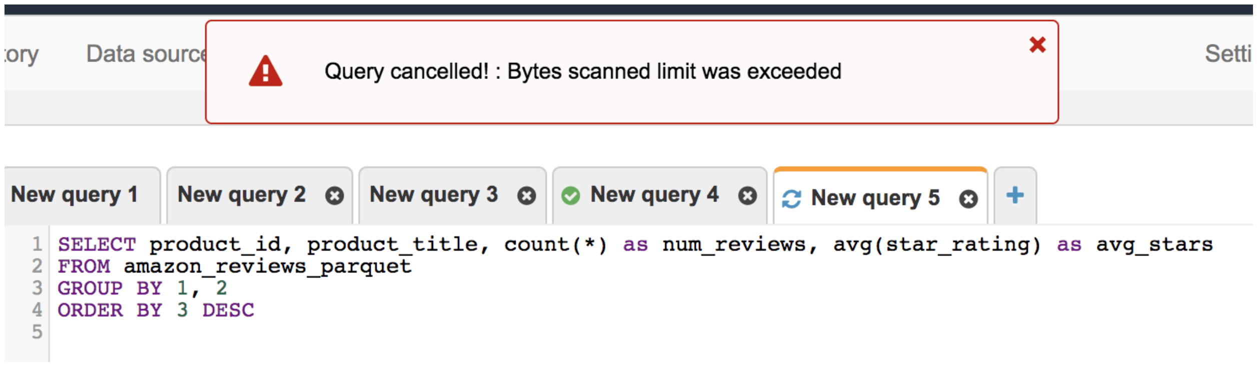
Action: If the query exceeds the limit, it will be cancelled.

5. Go back to query editor and select workgroup and switch to this new group. You will notice, it doesn't carry over the query history as workgroups separate out query execution and query history.
6. Execute following query. Query should run successfully and scan 215 MB. Notice we have filter on partition column and this is parquet formatted table

SELECT product_id, product_title, count(*) as num_reviews, avg(star_rating) as avg_stars
FROM amazon_reviews_parquet where product_category='Toys'
GROUP BY 1, 2
ORDER BY 3 DESC
(Run time: 8.05 seconds, Data scanned: 215.04 MB)

7. Execute following query without filter on partition column. You should get error message "Bytes Scanned limit was exceeded"

SELECT product_id, product_title, count(*) as num_reviews, avg(star_rating) as avg_stars
FROM amazon_reviews_parquet
GROUP BY 1, 2
ORDER BY 3 DESC



Query cancelled! : Bytes scanned limit was exceeded

```
SELECT product_id, product_title, count(*) as num_reviews, avg(star_rating) as avg_stars
FROM amazon_reviews_parquet
GROUP BY 1, 2
ORDER BY 3 DESC
```

Above steps shows control limits on per-query execution within a workgroup. You can also define workgroup-wide data usage control limit.