# Data Quality as part of a Data Catalogue

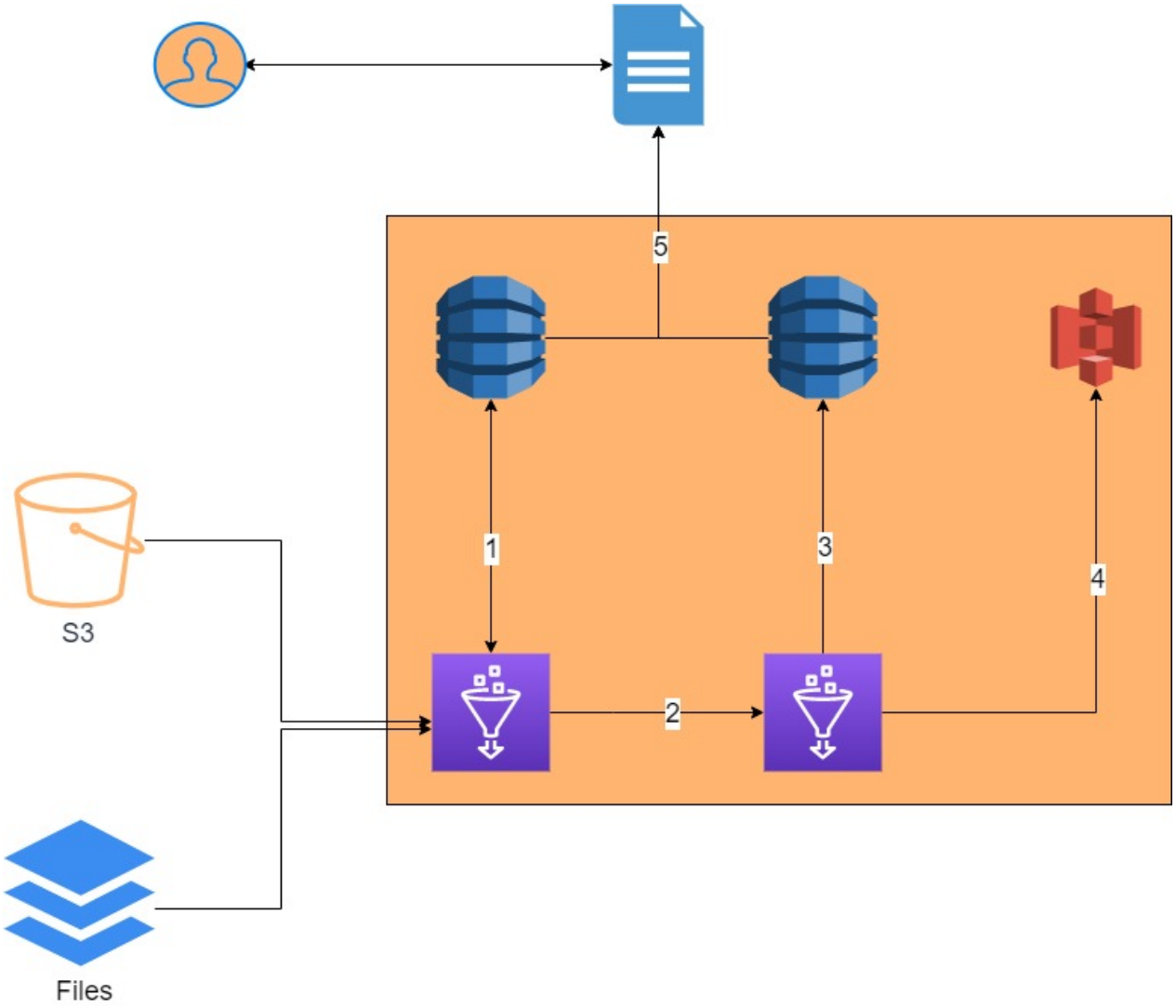
As data grows it is very important for modern organisations to understand what data exists and the data quality in their data stores (such as S3). With this artefact we will demonstrate how you can discover what data exists in your S3 buckets and the data quality in an automated way. This enables the creation of data lakes with trusted data without having to manually check what exists and the integrity of your data.

**Solution Architecture**

The solution is based on Amazon's Deequ ( https://aws.amazon.com/blogs/big-data/test-data-quality-at-scale-with-deequ/ ). Deequ is implemented on top fo Apache Spark and is designed to scale with large datasets that typically live in a distributed filesystem or a data warehouse. Deequ works on tabular data, e.g., CSV files, database tables, logs, flattened json files, basically anything that you can fit into a Spark dataframe.

This solution is based on the following approach:

1) Scan a given bucket and get the names of available files and store it to a DynamoDB table
2) A Glue job that grabs the values from step one and runs predefined data quality checks against the available files.
3) Results are stored to a DynamoDB table so they can be consumed by a data catalogue or any reporting tool
4) Results are also stored on S3 as we will utilize them on later releases as part of lineage report
5) Results are displayed into a central portal where users can also define the data quality metrics.




- **Pre-requisites**

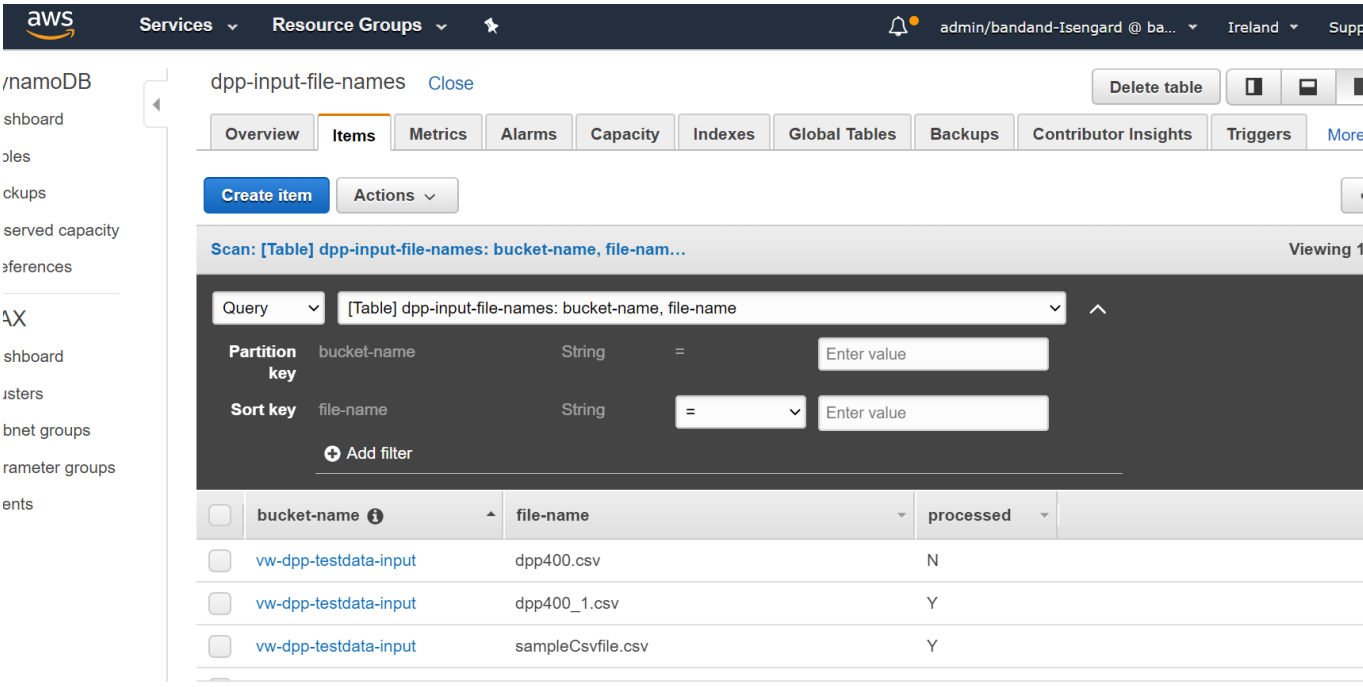
➢ **DynamoDB :**

Before creation of the Glue job, we create 2 DynamoDB tables. The idea is after the new files arrive to S3 bucket, the glue job needs to extract the file names and store in a DynamoDB input table (*dpp-input-file-names*) with processing status = N. After the job processed the files the metric result will be stored to another DynamoDB output table table (*dpp-output-metrics-result*) and it will update the file processed status as Y to input table.
The details of keys and sample screen print of these tables are mentioned below.

a) Input table - “dpp-input-file-names” to store the bucket name, file names and processed status. Before processing the file the processed status would be = N. After processing the file, the status would be updated to Y.

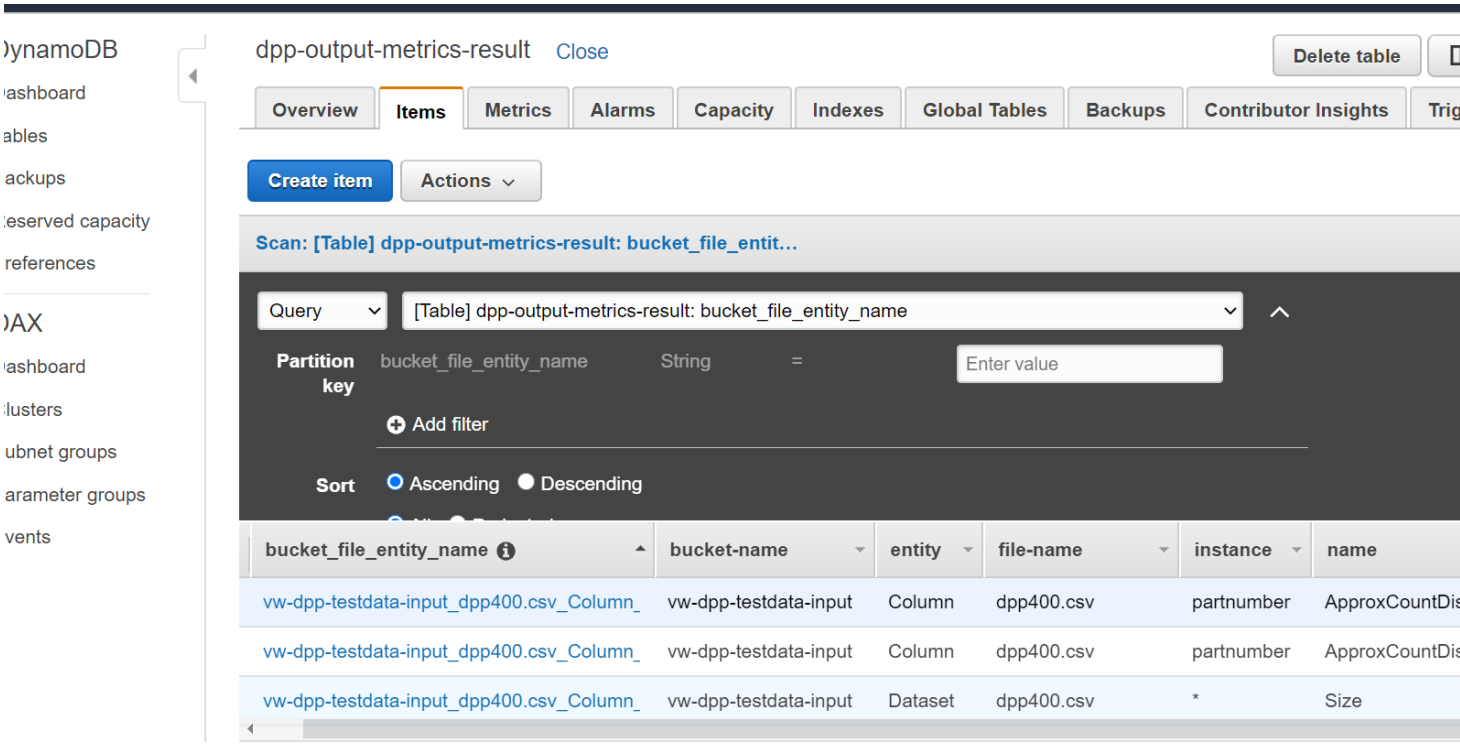
| Table name | dpp-input-file-names |
|---|---|
| Primary partition key | bucket-name (String) |

| | |
|---|---|
| Primary sort key | file-name (String) |




b) Output table - “dpp-output-metrics-result” to store the metric result for each of the file. Here the primary key has been created by combination of the business keys to make it unique - bucketName and FileName and entity and name of the attribute that we are measuring.

| | |
|---|---|
| Table name | dpp-output-metrics-result |
| Primary partition key | bucket_file_entity_name (String) |




- **S3 Buckets :**

S3 bucket names are globally unique and these buckets are already created. So you may need to create your own buckets in your accounts with different names. Accordingly you need to replace the bucket name (with your bucket name) in the scala scripts. You need to create the following buckets:

1) Input bucket: here you will store the data that you will use to run your DQ against. You can also store your dependent jars here ( 1.04 is the latest version – in this solution we use version 1.01 https://mvnrepository.com/artifact/com.amazon.deequ/deequ/1.0.1)

```
50
51  // read the file names from s3 bucket and store in a dataframe
52
53      val bucket = "achionis-deequ-testdata-input"
54      // val prefix = "input1" // In case prefix is required
55      val s3Client = new AmazonS3Client()
```

2) Output bucket: this is used to store the output data of the report. This is not necessary but we want to have this option as those outputs can be later used for lineage reports between files that have been scanned by this DQ solution.

```
212
213  // Additinally write metrics result on s3 as well ( to be seen via A
214
215  var opS3Path = "s3n://" + "achionis-deequ-testdata-output" + "/" +
216
```

- **Glue Crawler**

Create a crawler pointing at the output files on the output bucket as per below:

Crawlers > deequ output

Run crawler | Edit

| | |
|---|---|
| **Name** | deequ output |
| **Description** | |
| **Create a single schema for each S3 path** | false |
| **Security configuration** | |
| **Tags** | - |
| **State** | Ready |
| **Schedule** | |
| **Last updated** | Tue Sep 08 16:16:33 GMT+100 2020 |
| **Date created** | Tue Sep 08 16:16:33 GMT+100 2020 |
| **Database** | deequ_output |
| **Service role** | Glue_Admin |
| **Selected classifiers** | |
| **Data store** | S3 |
| **Include path** | s3://achionis-deequ-testdata-output/dpp400.csv/ |
| **Connection** | |
| **Exclude patterns** | |

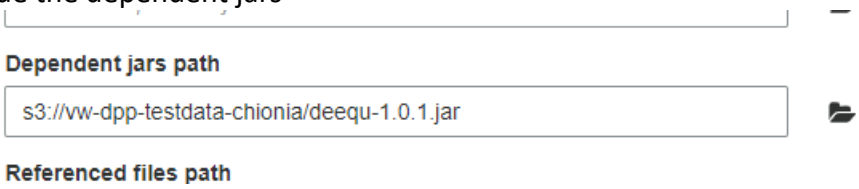
Configuration options

| | |
|---|---|
| **Schema updates in the data store** | Update the table definition in the data catalog. |
| **Object deletion in the data store** | Mark the table as deprecated in the data catalog. |

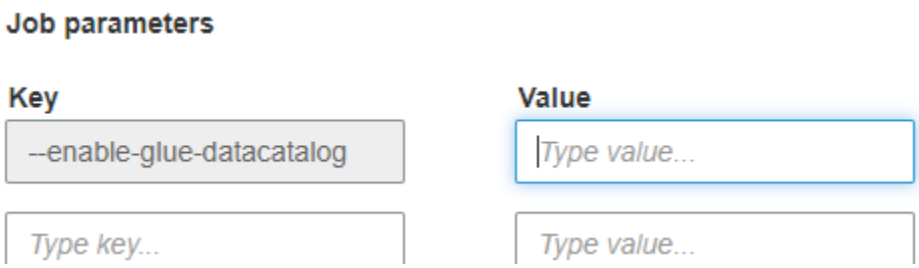
- **Glue Job**

Configuration of Glue Job

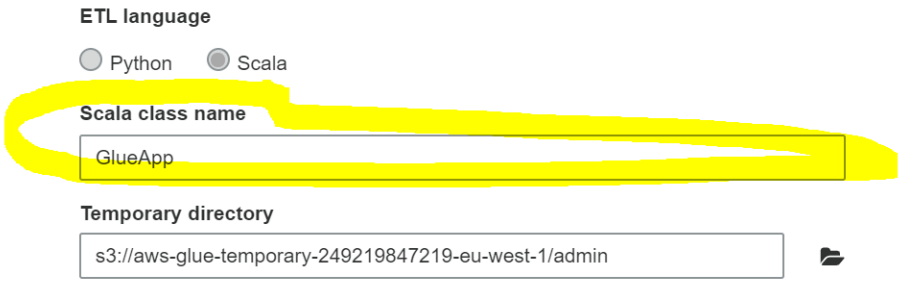
- Use Spark 2.4 , Scala 2 (glue 1.0)
- The role chosen while creation of Glue job should have access to S3 and DynamoDB for this job AmazonDynamoDBFullAccess, AmazonS3FullAccess and AWSGlueServiceRole access were chosen for the IAM role.
- Provide the dependent jars



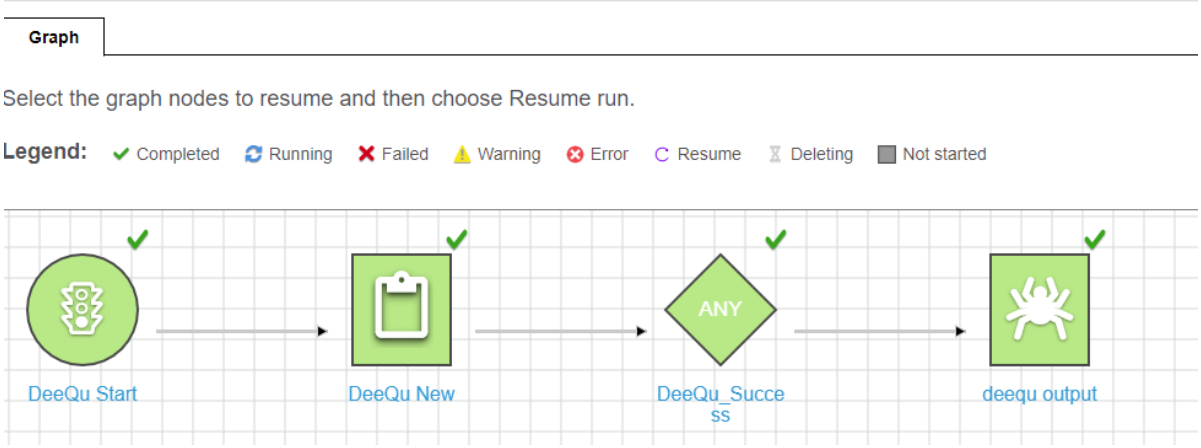

- Enable Glue Catalog for Hive Metastore



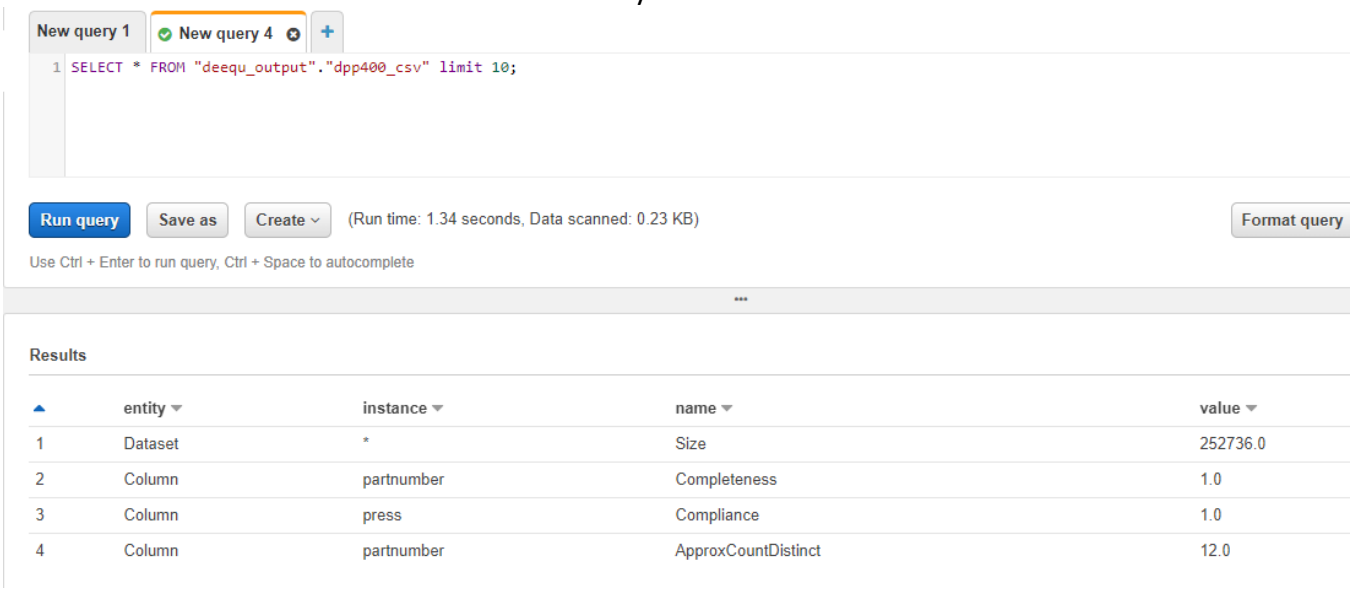

- Scala class name as - GlueApp

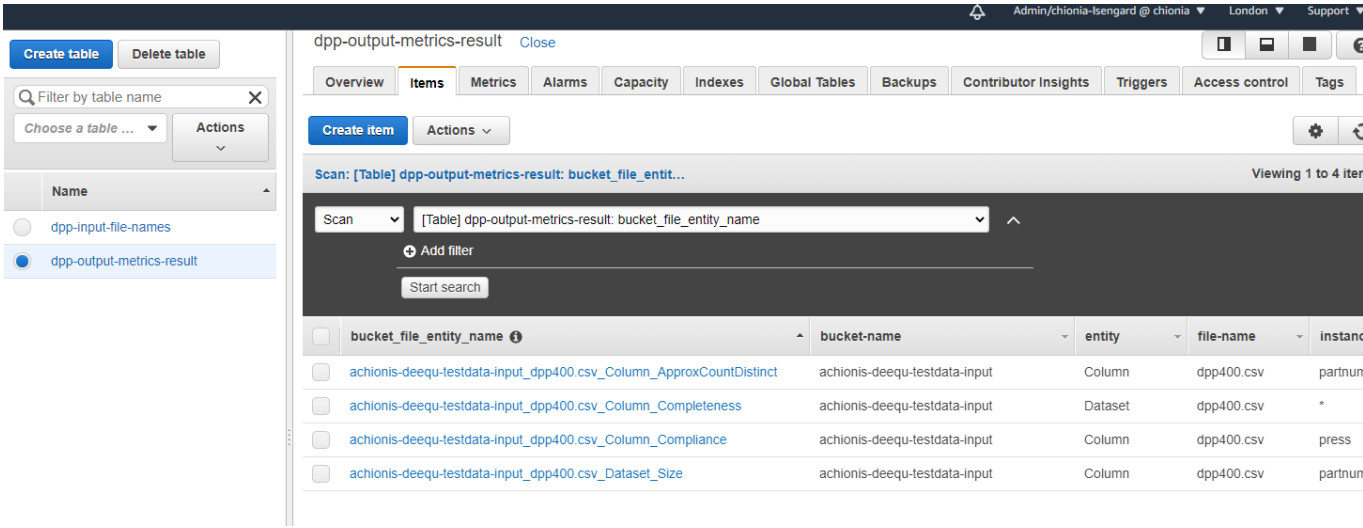



- Scala script changes as follows per this script : https://github.com/angeloschionis/Ingestion-Pipeline-Deequ/blob/master/DeeQuAnalysis.scala
- Then you can create a workflow that will run the Glue Job and Crawler that will populate an Athena table to allow any reporting need using this approach




- You can see the results both in Athena and DynamoDB




Admin/chionia-lsengard @ chionia
London
Support
Create table
Delete table
Filter by table name
Choose a table ...
Actions
Name
dpp-input-file-names
dpp-output-metrics-result
dpp-output-metrics-result
Close
Overview
Items
Metrics
Alarms
Capacity
Indexes
Global Tables
Backups
Contributor Insights
Triggers
Access control
Tags
Create item
Actions
Scan: [Table] dpp-output-metrics-result: bucket_file_entit...
Viewing 1 to 4 iter
Scan
[Table] dpp-output-metrics-result: bucket_file_entity_name
Add filter
Start search
bucket_file_entity_name
bucket-name
entity
file-name
instanc
achionis-deequ-testdata-input_dpp400.csv_Column_ApproxCountDistinct
achionis-deequ-testdata-input
Column
dpp400.csv
partnun
achionis-deequ-testdata-input_dpp400.csv_Column_Completeness
achionis-deequ-testdata-input
Dataset
dpp400.csv
*
achionis-deequ-testdata-input_dpp400.csv_Column_Compliance
achionis-deequ-testdata-input
Column
dpp400.csv
press
achionis-deequ-testdata-input_dpp400.csv_Dataset_Size
achionis-deequ-testdata-input
Column
dpp400.csv
partnun