Build your own data lake with AWS Glue and Amazon Athena

Damon Cortesi
@dacort

2 May 2019
Hi ♡ I’m

Damon Cortesi
A Big Data Architect

I work at AWS on EMR, Athena, Glue, and Lake Formation. On any given day, I can be found:

- Crafting presentations for both technical and executive audiences
- Debugging open source Hive, HBase, and Spark code
- Building proof of concepts and example architectures

© 2019, Amazon Web Services, Inc. or its Affiliates.
Intro to Data Lakes
There is **more data** than people think

<table>
<thead>
<tr>
<th>Data</th>
<th>Data platforms need to</th>
</tr>
</thead>
<tbody>
<tr>
<td>grows</td>
<td>live for</td>
</tr>
<tr>
<td>&gt;10x every 5 years</td>
<td>15 years</td>
</tr>
<tr>
<td></td>
<td>scale 1,000x</td>
</tr>
</tbody>
</table>

*IDC, Data Age 20215: The Evolution of Data to Life-Critical Don’t Focus on Big Data, Focus on the Data That’s Big, April 2017.*
There are more people accessing data

And more requirements for making data available

Data Scientists  Business Users
Analysts  Applications

Secure  Real time
Flexible  Scalable
# AWS databases and analytics

Broad and deep portfolio, built for builders

## Business Intelligence & Machine Learning

<table>
<thead>
<tr>
<th>Databases</th>
<th>Analytics</th>
<th>Blockchain</th>
<th>Data Lake</th>
<th>Data Movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon QuickSight</td>
<td>Amazon Redshift</td>
<td>AWS Glue</td>
<td>S3/Amazon Glacier</td>
<td>30+ solutions</td>
</tr>
<tr>
<td>Amazon SageMaker</td>
<td>Amazon EMR</td>
<td>ETL &amp; Data Catalog</td>
<td>Lake Formation</td>
<td>20+ Data lake solutions</td>
</tr>
<tr>
<td>Amazon Comprehend</td>
<td>Amazon Elasticsearch service</td>
<td></td>
<td>Data Lakes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Athena Interactive analytics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kinesis Analytics Real-time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AWS DeepLens</td>
<td>Amazon Elasticsearch service</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Operational Analytics</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Databases

- QLDB (NEW)
  - Ledger Database
- ElastiCache (NEW)
  - Redis, Memcached
- Aurora (NEW)
  - MySQL, PostgreSQL
- RDS (NEW)
  - MySQL, PostgreSQL, MariaDB, Oracle, SQL Server
  - RDS on VMware

## Analytics

- Neptune Graph
- DynamoDB (NEW)
  - Key value, Document
- Amazon Redshift
  - Data warehousing
- Amazon EMR
  - Hadoop + Spark
- Amazon Elasticsearch service
  - Operational Analytics

## Blockchain

- Amazon Glue
  - Managed Blockchain
  - Blockchain Templates

## Data Lake

- S3/Amazon Glacier (NEW)
- Lake Formation
  - Data Lakes

## Data Movement

- Database Migration Service
- Snowball
- Snowmobile
- Kinesis Data Firehose
- Kinesis Data Streams
- Data Pipeline
- Direct Connect

© 2019, Amazon Web Services, Inc. or its Affiliates.
A data lake is a centralized repository that allows you to store all your structured and unstructured data at any scale.
Damon’s data lake is a centralized repository of various quantified self and other personal stats including open Chrome Tabs, unread email count, GitHub repository stats and more
### AWS databases and analytics

**Broad and deep portfolio, built for builders**

<table>
<thead>
<tr>
<th>Business Intelligence &amp; Machine Learning</th>
<th>AWS Marketplace</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Databases</strong></td>
<td>250+ solutions</td>
</tr>
<tr>
<td>QLDB - Ledger Database</td>
<td>730+ Database solutions</td>
</tr>
<tr>
<td>ElastiCache - Redis, Memcached</td>
<td>600+ Analytics solutions</td>
</tr>
<tr>
<td>Aurora - MySQL, PostgreSQL</td>
<td>25+ Blockchain solutions</td>
</tr>
<tr>
<td>RDS - MySQL, PostgreSQL, MariaDB, Oracle, SQL Server</td>
<td>30+ Data lake solutions</td>
</tr>
<tr>
<td>S3/Amazon Glacier</td>
<td>20+ Data lake solutions</td>
</tr>
<tr>
<td>Lake Formation Data Lakes</td>
<td></td>
</tr>
<tr>
<td><strong>Analytics</strong></td>
<td></td>
</tr>
<tr>
<td>Amazon Redshift - Data warehousing</td>
<td></td>
</tr>
<tr>
<td>Amazon EMR - Hadoop + Spark</td>
<td></td>
</tr>
<tr>
<td>Athena - Interactive analytics</td>
<td></td>
</tr>
<tr>
<td>Amazon Elasticsearch service</td>
<td></td>
</tr>
<tr>
<td>Kinesis Analytics - Real-time</td>
<td></td>
</tr>
<tr>
<td><strong>Blockchain</strong></td>
<td></td>
</tr>
<tr>
<td>Managed Blockchain</td>
<td></td>
</tr>
<tr>
<td>Blockchain Templates</td>
<td></td>
</tr>
<tr>
<td><strong>Data Lake</strong></td>
<td></td>
</tr>
<tr>
<td>AWS Glue - ETL &amp; Data Catalog</td>
<td></td>
</tr>
<tr>
<td><strong>Data Movement</strong></td>
<td></td>
</tr>
<tr>
<td>Database Migration Service</td>
<td>Snowball</td>
</tr>
</tbody>
</table>

© 2019, Amazon Web Services, Inc. or its Affiliates.
Data lake with AWS Glue

Data

Move

Data Lake

Analyze

AWS Glue Data Catalog

Crawlers

Amazon S3 (Raw data) → AWS Glue → Amazon S3 (Staging data) → AWS Glue → Amazon S3 (Processed data)

Amazon Athena

Amazon QuickSight

Amazon Redshift

Amazon SageMaker

Amazon EMR
Data Ingestion
Amazon Kinesis Data Streams

Elastic

Real-time

Highly Scalable

Stream Analytics
with Kinesis Data Analytics
Amazon Kinesis Data Firehose

Serverless

Data Transforms

Near real-time loading to destination

Integrated
New! as of 12\textsuperscript{th} Feb

- Support for custom S3 prefix
Productivity Publisher

unread_count() {
  osascript -e 'tell application "Microsoft Outlook"
    unread count of folder "Inbox" of default account
  end tell'
}

chrome_tabs() {
  osascript -e 'tell application "Google Chrome" to count every tab of every window'
}

iterm_tabs() {
  osascript -e 'tell application "iTerm" to count every tab of every window'
}

UNREAD_COUNT=$(unread_count)
CURRENT_TIME=$(date -u +%Y-%m-%dT%H:%M:%SZ)
echo "Unread emails: \${UNREAD_COUNT}"
aws kinesis put-record --stream-name \${STREAM_NAME} --data '{
  "event": "outlook_unread", "type": "gauge",
  "value": \"\${UNREAD_COUNT}\", "ts": \"\${CURRENT_TIME}\"
}' --partition-key 1
Kinesis Firehose to S3

Amazon S3 destination

- **S3 bucket**: datalake-bucket
- **Prefix**: raw/life/year={timestamp:yyyy}/month={timestamp:MM}/day={timestamp:dd}/hour={timestamp:HH}/
- **Error prefix**: kinesisErrors/life/year={timestamp:yyyy}/month={timestamp:MM}/day={timestamp:dd}/hour={timestamp:HH}/[firehose:error-output-type]
- **Buffer conditions**: 128 MB or 900 seconds
- **Compression**: GZIP
- **Encryption**: Disabled
## Raw Data in S3

```
aws s3 ls s3://datalake-bucket/raw/life/year=2019/month=04/day=01/ --recursive | head
```

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Size</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-04-01</td>
<td>07:00:33</td>
<td>311</td>
<td>hour=05/dcortesi-lifestream-1-2019-04-01-05-45-29-cb63ffad-cf1c-4157-bb41-006e9d31bb6c.gz</td>
</tr>
<tr>
<td>2019-04-01</td>
<td>07:31:53</td>
<td>349</td>
<td>hour=06/dcortesi-lifestream-1-2019-04-01-06-16-50-e71c0155-8c3d-4c5b-8724-e1d7eb82b93a.gz</td>
</tr>
<tr>
<td>2019-04-01</td>
<td>08:03:22</td>
<td>307</td>
<td>hour=06/dcortesi-lifestream-1-2019-04-01-06-48-18-8f9c8f63-adf9-4ecb-b243-10e47c238c78.gz</td>
</tr>
<tr>
<td>2019-04-01</td>
<td>08:19:01</td>
<td>145</td>
<td>hour=07/dcortesi-lifestream-1-2019-04-01-07-03-59-c548c4c4-d233-4900-a7c6-236de3ba595d.gz</td>
</tr>
<tr>
<td>2019-04-01</td>
<td>16:07:15</td>
<td>345</td>
<td>hour=14/dcortesi-lifestream-1-2019-04-01-14-52-14-06a9b781-1ca4-448c-9ea8-f2f077a42265.gz</td>
</tr>
</tbody>
</table>
AWS Glue ETL

Serverless

Batch operations

Many sources

Apache Spark
Python and Scala

Python Shell New!
GitHub Traffic Stats

github_repos = ['dacort/athena-query-stats', 'dacort/damons-data-lake']
traffic_endpoints = ['popular/referrers', 'popular/paths', 'views', 'clones']

for repo in github_repos:
    for endpoint in traffic_endpoints:
        url = 'https://api.github.com/repos/' + repo + '/traffic/' + endpoint
        headers = {'Authorization': 'token ' + get_secret()}  # Use AWS Secrets Manager
        r = requests.get(url, headers=headers)
        if(r.ok):
            events = json.loads(r.text or r.content)
            # Only write out the file if we got any useful data
            if events:
                today = datetime.datetime.now().strftime('%Y-%m-%d')
                s3_key = '%s/%s/%s/%s.json' % (get_job_arg('prefix'),
                                               endpoint.replace('popular/', 'traffic/'),
                                               repo, today)
                save_results(get_job_arg('bucket'), s3_key, events)
CloudFront Logs

#Version: 1.0
#Fields: date time x-edge-location sc-bytes c-ip cs-method cs(Host) cs-uri-stem sc-status cs(Referer) cs(User-Agent) cs-uri-query cs(Cookie) x-edge-result-type x-edge-request-id x-host-header cs-protocol cs-bytes time-taken x-forwarded-for ssl-protocol ssl-cipher x-edge-response-result-type cs-protocol-version fle-status fle-encrypted-fields

2019-04-18 09:13:43 LAX3-C3 3439 10.0.0.1 GET d2tesbao5njk9n.cloudfront.net / 200 - Mozilla/5.0%2520(Macintosh;%2520Intel%2520%2520Mac%2520OS%2520X%252010_12_6)%2520AppleWebKit/537.36%2520(KHTML,%2520like%2520Gecko)%2520Chrome/73.0.3683.103%2520Safari/537.36 TUGAy9bu6cA_c0obQt_pne7A4Q_25Q== dacort.dev https 255 0.169 - TLSv1.2 ECDHE-RSA-AES128-GCM-SHA256 Miss HTTP/2.0 - -

2019-04-18 09:13:43 LAX3-C3 91094 10.0.0.1 GET d2tesbao5njk9n.cloudfront.net /assets/images/ab-img.png 200 https://dacort.dev/ Mozilla/5.0%2520(Macintosh;%2520Intel%2520%2520Mac%2520OS%2520X%252010_12_6)%2520AppleWebKit/537.36%2520(KHTML,%2520like%2520Gecko)%2520Chrome/73.0.3683.103%2520Safari/537.36 TUmpSnnw4yCP77em-u6Hds9hHaYftx_JOSo8WkI59qwNY-NVHShXag== dacort.dev https 42 0.123 - TLSv1.2 ECDHE-RSA-AES128-GCM-SHA256 Miss HTTP/2.0 - -

bucket-name.s3.amazonaws.com/optional-prefix/distribution-ID.YYYY-MM-DD-HH.unique-ID.gz
bucket-name.s3.amazonaws.com/optional-prefix/YYYY-mm-DD-HH-MM-SS-UniqueString
Fairly common problem

- Unstructured
- Multiple log types
- Different format
- Different compression
- One solution – embed parsing/conversion code in common library
  [github.com/awslabs/athena-glue-service-logs](https://github.com/awslabs/athena-glue-service-logs)

```python
from athena_glue_service_logs.job import JobRunner

job_run = JobRunner(service_name='s3_access')
job_run.convert_and_partition()
```
Configure with Job Parameters

```
{
    "Job": {
        "Command": {
            "Name": "glueetl",
            "ScriptLocation": "s3://bucket/scripts/s3_access_job.py"
        },
        "DefaultArguments": {
            "--extra-py-files": "s3://bucket/athena_glue_converter_latest.zip",
            "--s3_source_location": "s3://bucket/service_logs/s3_access/",
            "--raw_database_name": "awslogs_raw",
            "--raw_table_name": "s3_access",
            "--s3_converted_target": "s3://bucket/converted/s3_access",
            "--converted_database_name": "awslogs",
            "--converted_table_name": "s3_access",
            "--job-language": "python",
            "--job-bookmark-option": "job-bookmark-enable",
            "--TempDir": "s3://bucket/tmp"
        }
    }
}
```
Run every hour

Trigger properties
- Name: dcortesi-log-converter
- Tags: -
- Trigger type: Schedule
- Schedule: At 05 minutes past the hour

Jobs to start
- Jobs: dcortesi_dl_CloudFront_LogMaster_v5.3.3_dev, dcortesi_dl_S3Access_LogMaster_v5.3.4_dev
Parquet File Format

Columnar format is optimized for analytics.

Row group meta data allows Parquet reader to skip portions of, or all files.

Column meta-data allows for pre-aggregation.
## Results

<table>
<thead>
<tr>
<th>Conversion</th>
<th>File format</th>
<th>Data Scanned</th>
<th>Run time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT count(*) FROM s3_access_raw</td>
<td>text</td>
<td>7.1 GB</td>
<td>81 seconds</td>
</tr>
<tr>
<td>SELECT count(*) FROM s3_access_optimized</td>
<td>snappy parquet</td>
<td>0 KB</td>
<td>3.7 seconds</td>
</tr>
<tr>
<td>Speedup</td>
<td></td>
<td></td>
<td>22x faster</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conversion</th>
<th>File format</th>
<th>Data Scanned</th>
<th>Run time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT count(DISTINCT key) FROM s3_access_raw</td>
<td>text</td>
<td>7.1 GB</td>
<td>78 seconds</td>
</tr>
<tr>
<td>SELECT count(DISTINCT key) FROM s3_access_optimized</td>
<td>snappy parquet</td>
<td>91 MB</td>
<td>4.9 seconds</td>
</tr>
<tr>
<td>Speedup</td>
<td></td>
<td></td>
<td>16x faster</td>
</tr>
</tbody>
</table>
Data Discovery
So far we have...

- JSON traffic stats (uncompressed)
- JSON event streams (compressed)
- Tab-delimited service logs
- Regular Expression-based service logs
- Huge XML files
- Delivered via shell scripts, JavaScript, and email

- …not much different than the real world 😊
AWS Glue Data Catalog

United metadata repository across relational databases, Amazon RDS, Amazon Redshift, and Amazon S3.

Single searchable view into your data, no matter where it is stored.

Ability to automatically crawl and classify your data.

Augment technical metadata with business metadata for tables.

Manage access to data using Fine Grain Access Controls. Even finer with AWS Lake Formation.

Apache Hive metastore compatible and integrated with AWS Analytics services.
AWS Glue Crawlers

- Crawlers automatically build your Data Catalog and keep it in sync.
- Automatically discover new data, extracts schema definitions
  - Detect schema changes and version tables
  - Detect Hive style partitions on Amazon S3
- Built-in classifiers for popular types; custom classifiers using Grok expression
- Run ad hoc or on a schedule; serverless – only pay when crawler runs
Crawler Definition

```
{
  "Crawler": {
    "Name": "damons_data_lake",
    "Targets": {
      "S3Targets": [
        { "Path": "s3://bucket/raw/life/", "Exclusions": [] },
        { "Path": "s3://bucket/raw/github/clones/", "Exclusions": [] },
        { "Path": "s3://bucket/raw/github/traffic/", "Exclusions": [] },
        { "Path": "s3://bucket/raw/github/views/", "Exclusions": [] }
      ]
    },
    "DatabaseName": "dcortesi",
    "TablePrefix": "dl_",
    "Schedule": {
      "ScheduleExpression": "cron(20 0/1 * * ? *)",
      "State": "SCHEDULED"
    },
    "Configuration": "{\"Grouping\":{\"TableGroupingPolicy\":\"CombineCompatibleSchemas\"}}"
  }
}
```
AWS Lake Formation

Build a secure data lake in days

- Identify, ingest, clean, and transform data
- Enforce security policies across multiple services
- Gain and manage new insights
Analy(z|s)e!
Amazon Athena

Serverless

Interactive Performance

Open.Powerful.Standard
Built on Apache Presto

Pay per query
$0.005 per GB scanned
### Easily Query in Amazon Athena

#### Query
```sql
SELECT * FROM dl_life ORDER BY ts DESC LIMIT 10
```

#### Results
<table>
<thead>
<tr>
<th>event</th>
<th>type</th>
<th>value</th>
<th>ts</th>
<th>metadata</th>
</tr>
</thead>
<tbody>
<tr>
<td>item_tabs</td>
<td>gauge</td>
<td>6</td>
<td>2019-04-18T12:09:49Z</td>
<td></td>
</tr>
<tr>
<td>outlook_unread</td>
<td>gauge</td>
<td>18</td>
<td>2019-04-18T12:09:49Z</td>
<td></td>
</tr>
<tr>
<td>chrome_tabs</td>
<td>gauge</td>
<td>22</td>
<td>2019-04-18T12:09:49Z</td>
<td></td>
</tr>
<tr>
<td>outlook_unread</td>
<td>gauge</td>
<td>18</td>
<td>2019-04-18T12:08:39Z</td>
<td></td>
</tr>
<tr>
<td>chrome_tabs</td>
<td>gauge</td>
<td>22</td>
<td>2019-04-18T12:08:39Z</td>
<td></td>
</tr>
<tr>
<td>item_tabs</td>
<td>gauge</td>
<td>6</td>
<td>2019-04-18T12:08:39Z</td>
<td></td>
</tr>
<tr>
<td>chrome_tabs</td>
<td>gauge</td>
<td>22</td>
<td>2019-04-18T12:07:31Z</td>
<td></td>
</tr>
<tr>
<td>item_tabs</td>
<td>gauge</td>
<td>6</td>
<td>2019-04-18T12:07:31Z</td>
<td></td>
</tr>
<tr>
<td>outlook_unread</td>
<td>gauge</td>
<td>18</td>
<td>2019-04-18T12:07:31Z</td>
<td></td>
</tr>
<tr>
<td>clipboard_copy</td>
<td>counter</td>
<td>1</td>
<td>2019-04-18T12:06:32Z</td>
<td>{clipboard=[source=Google Chrome, destination=Google Chrome]}</td>
</tr>
</tbody>
</table>
Amazon EMR Notebooks in the Console

A managed analytics environment based on Jupyter Notebooks

- AWS Management Console for EMR
- EMR VPC
- EMR-managed notebook based on Jupyter notebook
- Amazon S3
- Auto saves notebook file to your S3 bucket
- Run queries on your remote EMR cluster
- Amazon EMR clusters
- Customer VPC
Apple Health Data

Data Exploration

In [3]:
   from pyspark.sql import SparkSession
   spark = SparkSession.builder.getOrCreate()
   df = spark.read.format('xml').options(rowTag='Record').load('s3

   Spark Job Progress

Data types for Motiv

In [4]:
   df.where("_sourceName = 'Motiv'").groupBy("_type").count().show

   Spark Job Progress

<table>
<thead>
<tr>
<th>__type</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKQuantityTypeIdentifierStepCount</td>
<td>234</td>
</tr>
<tr>
<td>HKQuantityTypeIdentifierHeartRate</td>
<td>18304</td>
</tr>
<tr>
<td>HKCategoryTypeIdentifierSleepAnalysis</td>
<td>100</td>
</tr>
</tbody>
</table>
Am I getting healthier?
Step Count (this month)
Step Count (this year)
Am I terrible at email?
Which projects should I focus on?
Is all my code from Stack Overflow?

<table>
<thead>
<tr>
<th>source</th>
<th>destination</th>
<th>copy_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Chrome</td>
<td>Google Chrome</td>
<td>562</td>
</tr>
<tr>
<td>iTerm2</td>
<td>iTerm2</td>
<td>148</td>
</tr>
<tr>
<td>Google Chrome</td>
<td>iTerm2</td>
<td>107</td>
</tr>
<tr>
<td>Code</td>
<td>Google Chrome</td>
<td>66</td>
</tr>
<tr>
<td>Code</td>
<td>Code</td>
<td>62</td>
</tr>
<tr>
<td>Google Chrome</td>
<td>Code</td>
<td>56</td>
</tr>
<tr>
<td>Code</td>
<td>iTerm2</td>
<td>41</td>
</tr>
<tr>
<td>Microsoft PowerPoint</td>
<td>Microsoft PowerPoint</td>
<td>37</td>
</tr>
<tr>
<td>iTerm2</td>
<td>Code</td>
<td>33</td>
</tr>
<tr>
<td>iTerm2</td>
<td>Google Chrome</td>
<td>32</td>
</tr>
</tbody>
</table>

SQL query:
```
SELECT metadata.clipboard.source, 
       metadata.clipboard.destination, 
       COUNT(*) AS copy_count 
FROM dl_life 
WHERE event = 'clipboard_copy' 
GROUP BY 1,2 
ORDER BY 3 DESC 
LIMIT 10
```
What interesting user agents are visiting my site?

### SQL Query

```sql
SELECT count(*) as cnt, url_decode(url_decode(useragent))
FROM "d cortesi"."dl_cloudfront_raw"
WHERE uri = '/' AND useragent NOT LIKE '%Mozilla%'
group by 2 order by 1 desc
limit 10;
```

### Results

<table>
<thead>
<tr>
<th>cnt</th>
<th>_col1</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>Go-http-client/1.1</td>
</tr>
<tr>
<td>11</td>
<td>Go-http-client/2.0</td>
</tr>
<tr>
<td>11</td>
<td>curl/7.58.0</td>
</tr>
<tr>
<td>5</td>
<td>Twitterbot/1.0</td>
</tr>
<tr>
<td>3</td>
<td>SMUrlExpander</td>
</tr>
<tr>
<td>3</td>
<td>Mediatoolkitbot (<a href="mailto:complaints@mediatoolkit.com">complaints@mediatoolkit.com</a>)</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Jetty/9.4.z-SNAPSHOT</td>
</tr>
<tr>
<td>2</td>
<td>Traackr.com</td>
</tr>
<tr>
<td>2</td>
<td>DAP/NetHTTP</td>
</tr>
</tbody>
</table>
How much data does CloudFront consume/generate?

```sql
SELECT
    SUM(CAST(bytes_sent as integer))/1024/1024 as uploadtotal_mib,
    SUM(CAST(object_size as integer))/1024/1024 as downloadtotal_mib,
    SUM(CAST(bytes_sent as integer) + CAST(object_size as integer))/1024/1024 AS total_mib
FROM dl_s3_access_optimus
WHERE user_agent = 'Amazon CloudFront'
LIMIT 10
```

(Run time: 3.9 seconds, Data scanned: 42.87 KB)

<table>
<thead>
<tr>
<th>uploadtotal_mib</th>
<th>downloadtotal_mib</th>
<th>total_mib</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>42</td>
<td>72</td>
</tr>
</tbody>
</table>
What does this all look like?

Sources
- Laptop
- GitHub API
- CloudFront
- Health

Kinesis Data Streams
- AWS Glue
- Python Shell
- Raw Bucket
- Me

Kinesis Data Firehose
- Crawlers
- AWS Glue
- Job
- Processed Bucket
- Me

Glue Data Catalog
- Crawlers
- Me

Amazon Athena
- Me

Amazon EMR Notebooks
- Me

github.com/dacort/damons-data-lake
Thank you!
Rate today’s session

Session page on conference website

O’Reilly Events App