Build a Data Lake with Amazon Web Services (AWS)

Damon Cortesi
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
Intro to Data Lakes
There is **more data than people think**

- Data grows >10x every 5 years
- Data platforms need to live for 15 years and scale 1,000x

---

*IDC, Data Age 2021: 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**
# AWS databases and analytics

Broad and deep portfolio, built for builders

## Databases
- **QLDB**: Ledger Database
- **ElastiCache**: Redis, Memcached
- **Aurora**: MySQL, PostgreSQL
- **RDS**: MySQL, PostgreSQL, MariaDB, Oracle, SQL Server

## Analytics
- **Amazon Redshift**: Data warehousing
- **Amazon EMR**: Hadoop + Spark
- **Kinesis Analytics**: Real-time
- **Amazon Elasticsearch service**: Operational Analytics

## Blockchain
- **Governance & Compliance**
- **Collaboration & Identity**
- **Data Compliance**
- **Supply Chain**
- **IoT**
- **Automotive**
- **Healthcare**
- **Retail**
- **Travel**
- **Gaming**
- **Sports & Entertainment**

## Data Lake
- **S3/Amazon Glacier**: ETL & Data Catalog

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

## AWS Marketplace
- **250+ solutions**
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>Databases</th>
<th>Analytics</th>
<th>Blockchain</th>
<th>Data Lake</th>
<th>Data Movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDS (MySQL, PostgreSQL, MariaDB, Oracle, SQL Server)</td>
<td>Amazon Redshift (Data warehousing)</td>
<td>Managed Blockchain</td>
<td>S3/Amazon Glacier</td>
<td>Database Migration Service</td>
</tr>
<tr>
<td>ElastiCache (Redis, Memcached)</td>
<td>Amazon EMR (Hadoop + Spark)</td>
<td>Blockchain Templates</td>
<td>Lake Formation (Data Lakes)</td>
<td>Snowball</td>
</tr>
<tr>
<td>Aurora (MySQL, PostgreSQL)</td>
<td>Athena (Interactive analytics)</td>
<td>25+ Blockchain solutions</td>
<td>AWS Glue (ETL &amp; Data Catalog)</td>
<td>Snowmobile</td>
</tr>
<tr>
<td>Neptune (Graph)</td>
<td>Kinesis Analytics (Real-time)</td>
<td>730+ Database solutions</td>
<td>AWS Glue (ETL &amp; Data Catalog)</td>
<td>Kinesis Data Firehose</td>
</tr>
<tr>
<td>DynamoDB (Key value, Document)</td>
<td>Amazon Elasticsearch service (Operational Analytics)</td>
<td>600+ Analytics solutions</td>
<td>AWS Glue (ETL &amp; Data Catalog)</td>
<td>Kinesis Data Streams</td>
</tr>
<tr>
<td>Timestream (Time Series)</td>
<td></td>
<td>20+ Data lake solutions</td>
<td></td>
<td>Data Pipeline</td>
</tr>
<tr>
<td>RDS on VMWare</td>
<td></td>
<td></td>
<td></td>
<td>Direct Connect</td>
</tr>
</tbody>
</table>

**AWS Marketplace**

- 250+ solutions
- 730+ Database solutions
- 600+ Analytics solutions
- 25+ Blockchain solutions
- 20+ Data lake solutions
- 30+ solutions

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

Data

AWS Direct Connect
AWS Snowball
Amazon Kinesis
AWS IoT Core
AWS Database Migration Service
AWS Storage Gateway

Move

Crawlers

Data Lake

AWS Glue Data Catalog

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

Analyze

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
Kinesis Data Streams to Amazon S3

Clients

Kinesis Data Streams → AWS Lambda Transformation → Kinesis Data Firehose

Prefix: raw/life/year={timestamp:yyyy}/month={timestamp:MM}/day={timestamp:dd}/
Buffer: Up to 128MB or 15 minutes

Aggregated JSON Data → Crawlers

Save as Parquet

Aggregated Parquet Data

Source backup

Amazon Athena
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 Data Firehose to Amazon S3

## Amazon S3 destination

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3 bucket</td>
<td>datalake-bucket</td>
</tr>
<tr>
<td>Prefix</td>
<td>raw/life/year={timestamp:yyyy}/month={timestamp:MM}/day={timestamp:dd}/hour={timestamp:HH}/</td>
</tr>
<tr>
<td>Error prefix</td>
<td>kinesisErrors/life/year={timestamp:yyyy}/month={timestamp:MM}/day={timestamp:dd}/hour={timestamp:HH}/(firehose:errors-error-output-type)</td>
</tr>
<tr>
<td>Buffer conditions</td>
<td>128 MB or 900 seconds</td>
</tr>
<tr>
<td>Compression</td>
<td>GZIP</td>
</tr>
<tr>
<td>Encryption</td>
<td>Disabled</td>
</tr>
</tbody>
</table>
### Raw Data in Amazon S3

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

<table>
<thead>
<tr>
<th>Time</th>
<th>Size</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-04-01 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 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 08:03:22</td>
<td>307</td>
<td>hour=06/dcortesi-lifestream-1-2019-04-01-06-48-18-8f9c8f63-ad9f-4ecb-b243-10e47c238c78.gz</td>
</tr>
<tr>
<td>2019-04-01 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 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>

© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
AWS Glue ETL

Serverless

Batch operations

Many sources

Apache Spark
Python and Scala

Python Shell
Third-party API to Amazon S3

3rd Party API

AWS Glue Python Shell

Incremental Exports

Crawlers

Data Catalog

Transformed Data

AWS Glue ETL

Amazon Redshift

Amazon Athena
GitHub Traffic Stats

github_repos = ['dacort/athena-query-stats', 'dacort/demons-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()}
        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)

Website Logs to Amazon S3

Commit Hook

AWS CodeBuild

AWS CodePipeline

Website Artifacts

Amazon CloudFront

Internet

AWS Glue Job

Amazon S3 Server Access and CloudFront logs

Data Catalog

Transformed Data

Amazon Athena
Amazon 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%20(Macintosh;%20Intel%20Mac%20OS%20X%2010_12_6)%20AppleWebKit/537.36%20(KHTML,%20like%20Gecko)%20Chrome/73.0.3683.103%20Safari/537.36 - - Miss U6zGSZXHb6peSeVwpRPJ8o0-TUGAy9bu6C_A_c0obQt_pne7A4Q_25Q== dacort.dev https 255 0.169 - 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

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%20(Macintosh;%20Intel%20Mac%20OS%20X%2010_12_6)%20AppleWebKit/537.36%20(KHTML,%20like%20Gecko)%20Chrome/73.0.3683.103%20Safari/537.36 - - Miss TUmSpSnw4yP77em-u6Hds9hHaYftx_JOSo8WkIS9qwNY-NVHShXag== dacort.dev https 42 0.123 - TLSv1.2 ECDHE-RSA-AES128-GCM-SHA256 Miss HTTP/2.0 - -
Amazon S3 Server Access Logs

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
  [link to a repository](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

```json
{
  "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",
      "--s3Converted_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

Unified 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

```json
{
    "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
Analyze!
Amazon Athena

Serverless

Interactive Performance

Open.Powerful.Standard
Built on Apache Presto

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

**New query 1**

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

**Run query** (Run time: 4.72 seconds, Data scanned: 174.4 KB)

**Database**

- `dcortesi`
- `dl_life`

**Tables (1)**

- **dl_life** (Partitioned)
  - `event` (string)
  - `type` (string)
  - `value` (string)
  - `ts` (string)
  - `metadata` (struct<clipboard.struct<source:string, destination:string>>)

**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_tab</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_tab</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_tab</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

users

AWS Management Console for EMR

EMR VPC

EMR-managed notebook based on Jupyter notebook

Run queries on your remote EMR cluster

Auto saves notebook file to your S3 bucket

Amazon S3

Amazon EMR clusters

Customer VPC

Spark

Amazon EMR

Amazon EMR

Amazon EMR

© 2019, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
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
```

Data types for Motiv

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

Spark Job Progress

```
+-----------------+-----+
|_type            |count|
+-----------------+-----+
|HKQuantityTypeIdentifierStepCount | 234 |
|HKQuantityTypeIdentifierHeartRate  | 18304|
|HKCategoryTypeIdentifierSleepAnalysis|100  |
+-----------------+-----+
```
Am I getting healthier?
Step Count (Apr 2019)
Step Count (Jan – Apr 2019)
Am I terrible at email?
Which projects should I focus on?

Views by Day and Repository

- awslabs/athena-adobe-datafeed-splitter
- awslabs/athena-glue-service-logs
- dacort/athena-query-stats
- dacort/demons-data-lake
- dacort/demo-code
- dacort/sample-code
- dacort/syslog-to-athena
What apps are responsible for my RSI?

(Repetitive strain injury)

<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:
```sql
SELECT
  clipboard source,
  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?

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

<table>
<thead>
<tr>
<th>cnt</th>
<th>_col1</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>Go-http-client/1.1</td>
</tr>
<tr>
<td>68</td>
<td>Go-http-client/2.0</td>
</tr>
<tr>
<td>24</td>
<td>curl/7.58.0</td>
</tr>
<tr>
<td>21</td>
<td>Twitterbot/1.0</td>
</tr>
<tr>
<td>20</td>
<td>python-requests/2.22.0</td>
</tr>
<tr>
<td>19</td>
<td>Java/1.8.0_212</td>
</tr>
<tr>
<td>14</td>
<td>python-requests/2.22.0</td>
</tr>
<tr>
<td>11</td>
<td>RandomSurfer/1.0 (<a href="https://random.surf/bot">https://random.surf/bot</a>)</td>
</tr>
<tr>
<td>10</td>
<td>Apache-HttpClient/4.5.2 (Java/1.8.0_171)</td>
</tr>
<tr>
<td>9</td>
<td>CheckMarkNetwork/1.0 (<a href="http://www.checkmarknetwork.com/spider.html">http://www.checkmarknetwork.com/spider.html</a>)</td>
</tr>
</tbody>
</table>

(Run time: 2.18 seconds, Data scanned: 247.11 KB)
What does this all look like?

Sources

- Laptop
- GitHub API
- CloudFront
- Health

Me

Kinesis Data Streams

Kinesis Data Firehose

AWS Glue Python Shell

Raw Bucket

AWS Glue Job

Processed Bucket

Crawlers

AWS Glue Data Catalog

Amazon Athena

Amazon EMR Notebooks

github.com/dacort/damons-data-lake
Thank you!

Be sure to check out these re:Invent sessions!

- AN202 - Turbocharge your Spark performance with Amazon EMR
- AN205 - What's new with Amazon Athena
- AN307 - Deep dive into Amazon Athena
- AN308 - Deep dive into running Apache Spark on Amazon EMR