Aggregating CloudTrail log files

A data pipeline for reprocessing files in a data lake to optimize query performance.

Keith Gregory
AWS Practice Lead, Chariot Solutions

CHARIOT SOLUTIONS

The Problem

Lots of small files in a data lake

Small Files == Poor Query Performance

It takes ~15ms to read a 1 KB file from S3 to EC2

It takes ~20ms to read 1 MB

This translates to 67 files per second, per reader

Real-world numbers counting CloudTrail events:

Raw CloudTrail logs (1,637,376 files):	3 minutes 15 seconds
Aggregated by date (684 files):	6.3 seconds

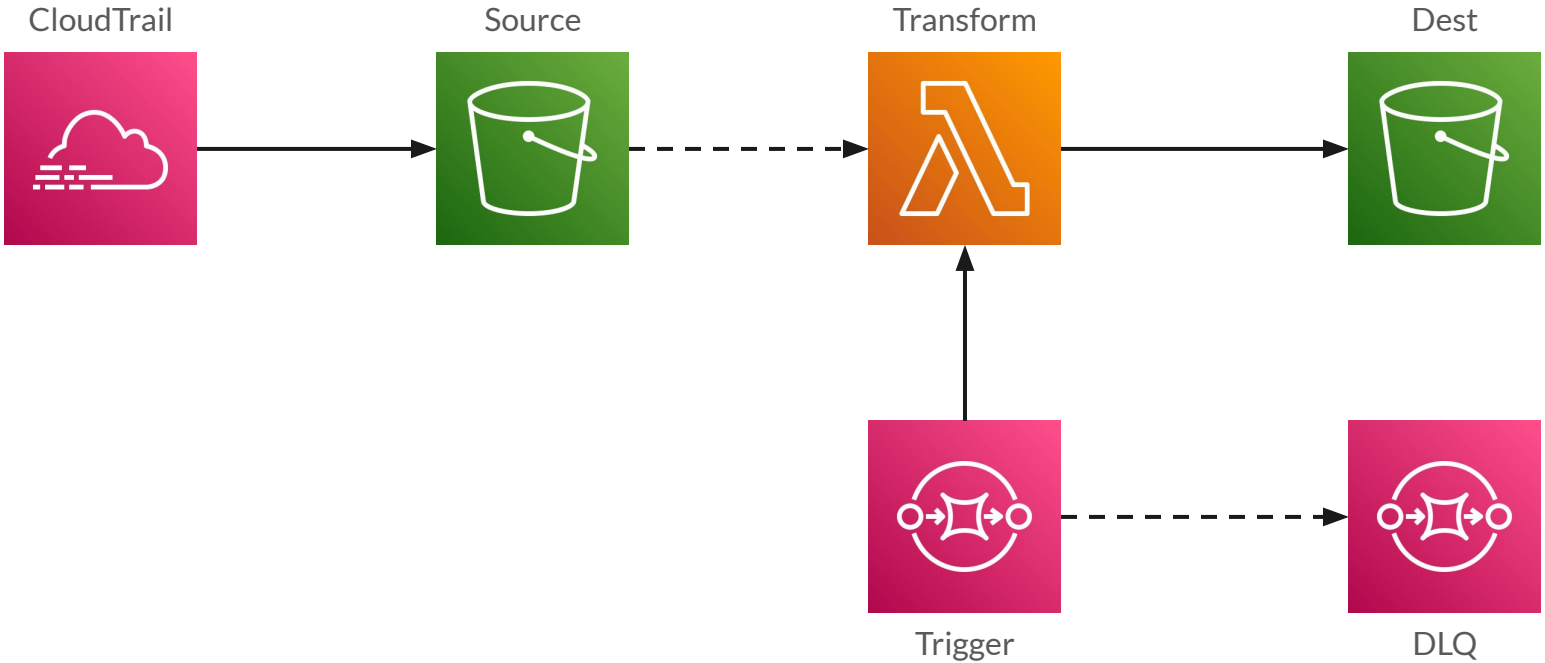
CloudTrail Writes Lots Of Small Files

- Approximately every 15 minutes, per account, per region
  - In Chariot's case, this is over 4,000 files per day
  - 4.6 million since we enabled CloudTrail
- These files often contain only a few events
  - In Chariot's case, 44% of files have 4 events or fewer (1k filesize)

The Solution

You know there had to be one, right?

A Lambda to Aggregate Files



Triggered from SQS

- Simple integration with Lambda
- Retry on failure, with persistent failures sent to dead-letter queue
- Supports both manual and programmatic invocation

Code Sample: Sending Messages to SQS

```
#!/usr/bin/env python3

import boto3
import json

from datetime import date, timedelta

QUEUE_URL = "https://sqs.us-east-1.amazonaws.com/123456789012/cloudtrail-aggregation-trigger"

client = boto3.client('sqs')

dd = date(2023, 12, 1)
while dd <= date(2023, 12, 31):
    msg = json.dumps({"month": dd.month, "day": dd.day, "year": dd.year})
    client.send_message(QueueUrl=QUEUE_URL, MessageBody=msg)
    dd += timedelta(days=1)
```

Scaling the Transform

SQS invokes concurrent Lambdas to process queue

Quickly scales to configured limit or account quota

This can crush a database server!

May cause starvation for other Lambda jobs!

List One Day's Files

CloudTrail has a complex naming scheme:

```
AWSLogs/o-x7q8b22ufe/123456789012/CloudTrail/us-east-1/2
024/01/05/123456789012_CloudTrail_us-east-1_20240105T113
0Z_sKZpDSwJukKLNIFI.json.gz
```

S3 isn't a filesystem

But `ListObjectsV2` can make it act like one, using repeated calls with successively longer prefixes

Code Sample: Retrieving Prefixes

```
# src_bucket = "com-chariotsolutions-cloudtrail"
# src_prefix = "AWSLogs/o-x7q8b22ufe/"


def retrieve_accounts():
    return retrieve_child_prefixes(src_prefix)


def retrieve_regions_for_account(account_id):
    return retrieve_child_prefixes(f"{src_prefix}{account_id}/CloudTrail/")


def retrieve_child_prefixes(parent_prefix):
    resp = s3_client.list_objects_v2(Bucket=src_bucket, Prefix=parent_prefix, Delimiter="/")
    result = []
    for prefix in [x['Prefix'] for x in resp.get('CommonPrefixes', [])]:
        trimmed = prefix.replace(parent_prefix, "").replace("/", "")
        result.append(trimmed)
    return result
```

Code Sample: Listing Files

```
# src_bucket = "com-chariotsolutions-cloudtrail"
# src_prefix = "AWSLogs/o-x7q8b22ufe/"


def retrieve_file_list_for_account_region_and_date(account_id, region, month, day, year):
    result = []
    prefix = f"{src_prefix}{account_id}/CloudTrail/{region}/{year:04d}/{month:02d}/{day:02d}/"
    req_args = {
        "Bucket": src_bucket,
        "Prefix": prefix
    }
    while True:
        resp = s3_client.list_objects_v2(**req_args)
        for item in resp.get('Contents', []):
            result.append(item['Key'])
        if not resp.get('IsTruncated'):
            return result
        req_args['ContinuationToken'] = resp['NextContinuationToken']
```

Aggregate Files In-Memory

Historically, Lambda provided only 512 MB disk

Still much easier to let the garbage collector reclaim memory

Code Sample: Aggregating Records

```
def retrieve_log_records(file_list):
    for file in file_list:
        parsed = json.loads(read_file(file))
        for rec in parsed['Records']:
            yield json.dumps(rec)

def aggregate_and_output(file_list, month, day, year, desired_size = 64 * 1024 * 1024):
    file_number = 0
    cur_recs = []
    cur_size = 0
    for rec in retrieve_log_records(file_list):
        cur_recs.append(rec)
        cur_size += len(rec)
        if cur_size >= desired_size:
            write_file(cur_recs, month, day, year, file_number)
            file_number += 1
            cur_recs = []
            cur_size = 0
    if len(cur_recs) > 0:
        write_file(cur_recs, month, day, year, file_number)
```

Code Sample: Writing Files

```
def write_file(records, month, day, year, file_number):
    out = io.StringIO()
    for rec in records:
        print(rec, file=out)
    data = out.getvalue().encode('utf-8')
    data = gzip.compress(data)
    key = f"{dst_prefix}{year:04d}/{month:02d}/{day:02d}/{file_number:06d}.ndjson.gz"
    logger.info(f"writing {len(records)} records to s3://{dst_bucket}/{key}")
    s3_client.put_object(Bucket=dst_bucket, Key=key, Body=data)
```

Resiliency

Pipelines fail, be prepared

Do or Do Not, There is No “Try”

- It’s better to fail than partially succeed
  - Don’t catch exceptions unless you know that you can handle them
- Fail loudly
  - Nobody looks at logs unless they know there’s a problem
  - But capture enough information to allow debugging!
- Make transformer idempotent: invoke many times, produce same output

CloudWatch Metrics / Alarms

- Built-in metrics:
  - Alarm if anything in dead-letter queue
  - Alarm if messages remain in trigger queue
- Custom metrics to report file count, number of records
  - Alarm when out of bounds, or if missing data
- Dashboards only useful if people look at them

Logging

INFO-level to report high-level activity

```
2024-01-23T22:39:03.789Z retrieving files for 2024-01-12
2024-01-23T22:39:16.743Z 4002 total files
2024-01-23T22:42:41.953Z writing 18698 records to
s3://example/cloudtrail/2024/01/12/000000.ndjson.gz
```

DEBUG-level for detailed actions

```
2024-01-23T22:39:27.112Z 284 files for account 123456789012,
region us-west-2
```

Configure logging level via environment variable

Performance

How slow is slow, and what can we do about it?

It Takes Four Minutes to Run!

Number of source files:	4,002
Number of records:	18,698
Time breakdown from logs:	
Select list of files:	12.954 sec
Read files and aggregate records:	205.501 sec
Total execution time:	218.455 sec

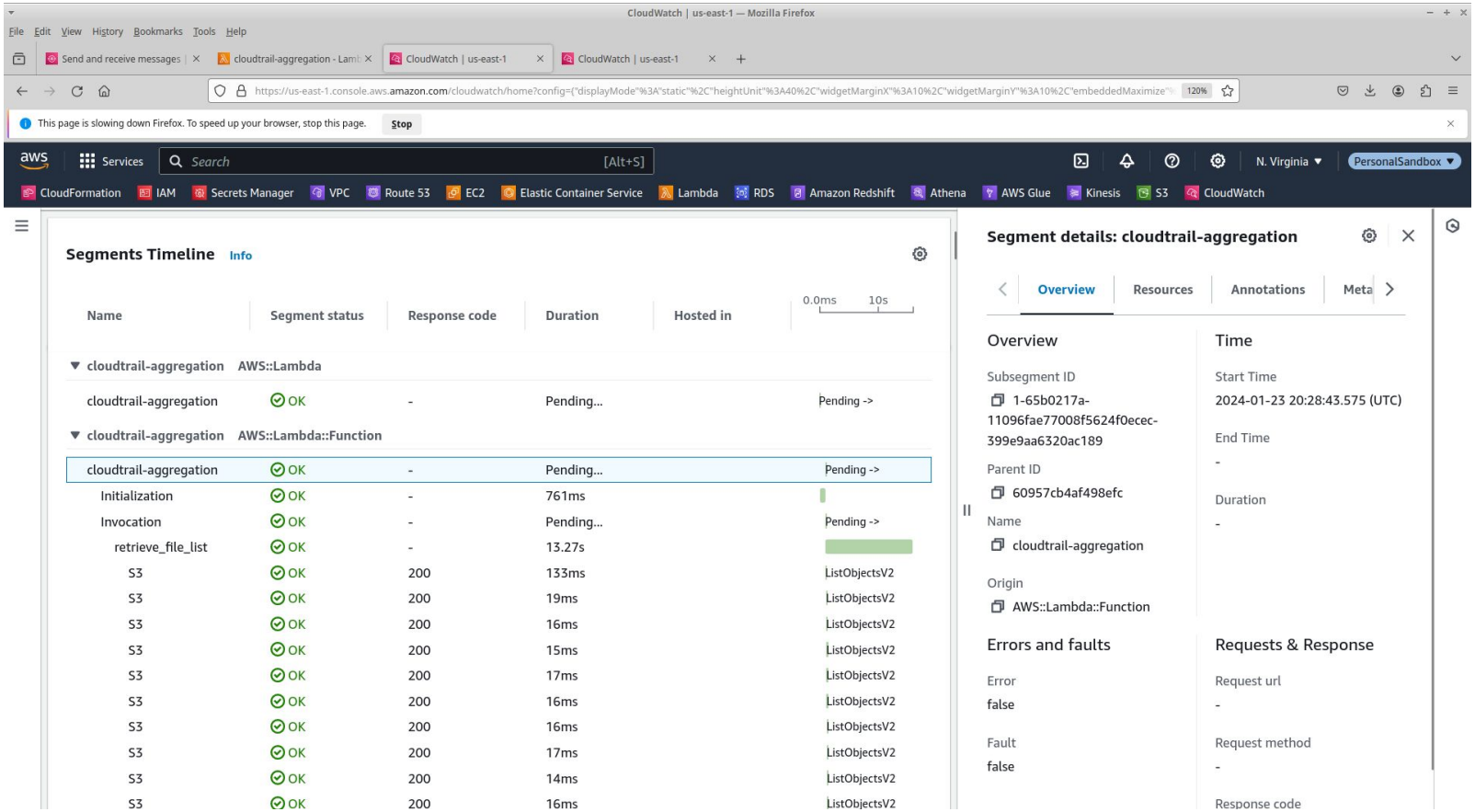
Will More Memory Help?

- Provisioned memory determines CPU
  - 1,769 MB == 1 vCPU
  - 128 MB (default for Python) == 1/14th vCPU
- This is primarily a problem for cold starts
  - Especially for scripts that load a lot of modules and/or perform CPU-intensive tasks at startup (including boto3 client creation)

Can X-Ray Give Us More Information?

- Distributed tracing service provided by AWS
  - Reports “trace segments” to central service
- Allows patching libraries (including boto3) to trace calls to external services
- Tightly integrated with Lambda
  - Custom segments requires the X-Ray SDK; attach as layer

X-Ray Isn't Designed For This!



How About the Python Profiler?

Tracing profiler built-in to the CPython runtime

Lambda throws when using `cProfile.run()`

Instead, must explicitly create, enable, disable:

```
profiler = cProfile.Profile()
profiler.enable()
aggregate_and_output(all_files, month, day, year)
profiler.disable()
pstats.Stats(profiler).sort_stats('tottime').print_stats()
```

Function Calls by Cumulative Time

```
31596718 function calls (31400363 primitive calls) in 252.702 seconds
Ordered by: cumulative time
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
1        0.043    0.043   252.764  252.764 /var/task/index.py:99(aggregate_and_output)
18699    0.170    0.000   248.831    0.013 /var/task/index.py:119(retrieve_log_records)
4002     0.061    0.000   247.582    0.062 /var/task/index.py:129(read_file)
4003     0.028    0.000   246.635    0.062 /opt/python/botocore/client.py:544(_api_call)
(a bunch of functions in the boto3 call chain)
8324   210.584    0.025   210.584    0.025 {method 'read' of '_ssl._SSLSocket' objects}
```

Is there an issue with buffer size?

Possible Implementation Changes

Multi-threading?

- Can give performance boost when IO overlaps with CPU
- Python GIL means that Amdahl's Law limits speedup
- Adds complexity: need to coordinate threads

Use a different language?

- A "faster" language won't reduce IO times
- But other languages have better support for concurrency

Is This a Problem That Needs Solving?

- This Lambda typically runs when nobody's watching
- 4 minutes, 512 MB == 120 GB/sec == $0.002
  - Run daily, that's $0.78 / year
- Scale horizontally to process an entire dataset
  - With 8 concurrent executions, a month of data takes < 15 minutes

But Wait, There's More

Because friends don't let friends use JSON in their data lake

Reasons to stop here

CloudTrail data is complex, changing, impossible to accurately represent in a fixed format like Parquet

A few thousand files is sufficiently performant

Each file is ~6 MB; too small for Parquet

A Second Aggregation Step

- We process data by month this time
  - Reads data produced by first step
  - Lambda would timeout if we gave it a whole month of raw data
- Question: do we need to retain that first aggregation?
  - Idempotency says the files *should not* be deleted by the Lambda
- Nested fields will be stored as JSON strings

PyArrow

- Python wrapper over native Arrow library
- Better-documented than the Java Parquet library
  - And it doesn't have hundreds of dependencies!
- API is designed for bulk activities
  - This means staging data as Python lists/dicts, then writing

Code Sample: Schema

```
SCHEMA = pa.schema([
    pa.field('eventID', pa.string()),
    pa.field('requestID', pa.string()),
    pa.field('sharedEventID', pa.string()),
    pa.field('eventTime', pa.timestamp('ms')),
    pa.field('eventName', pa.string()),
    pa.field('eventSource', pa.string()),
    pa.field('eventVersion', pa.string()),
    pa.field('awsRegion', pa.string()),
    pa.field('sourceIPAddress', pa.string()),
    pa.field('recipientAccountId', pa.string()),
    pa.field('userIdentity', pa.string()),
    pa.field('requestParameters', pa.string()),
    pa.field('responseElements', pa.string()),
    pa.field('additionalEventData', pa.string()),
    pa.field('resources', pa.string()),
])
```

Code Sample: Record Conversion

```
def transform_record(src_rec):
    rec = json.loads(src_rec)
    xformed = {
        'eventID': rec.get('eventID'),
        'requestID': rec.get('requestID'),
        'sharedEventID': rec.get('sharedEventID'),
        'eventTime': datetime.fromisoformat(rec.get('eventTime')),
        # ...
        'recipientAccountId': rec.get('recipientAccountId'),
    }
    for nested_key in ['userIdentity', 'requestParameters', 'responseElements',
                       'additionalEventData', 'resources']:
        if rec.get(nested_key):
            xformed[nested_key] = json.dumps(rec.get(nested_key))
        else:
            xformed[nested_key] = None
    return xformed
```

Code Sample: Writing the File

```
def write_file(records, month, year, file_number):
    s3_url = f"s3://{dst_bucket}/{dst_prefix}{year:04d}/{month:02d}/{file_number:06d}.parquet"
    logger.info(f"writing {len(records)} records to {s3_url}")
    table = pa.Table.from_pylist(records, schema=SCHEMA)
    pq.write_table(table, s3_url, compression='SNAPPY')
```

A “Big” Problem

PyArrow + NumPy > 50 MB

Layers to the rescue!

Must use two: NumPy is 23M, PyArrow is 39 MB

Fortunately, there aren’t any transitive dependencies

```
pip install -t python pyarrow
zip -r /tmp/pyarrow.zip python/pyarrow*
zip -r /tmp/numpy.zip python/numpy*
```

A Bigger Problem

PyArrow library is specific to Python version, processor architecture

If you get this wrong: `No module named 'pyarrow.lib'`

Solution #1: create using AWS Docker image

```
docker run -it --rm \
       --entrypoint /bin/bash -v /tmp:/build \
       amazon/aws-lambda-python:3.11
```

Solution #2: deploy as a Lambda Container Image

Questions?

I have answers, some of them might even be correct

Technology in the Service of Business.

Chariot Solutions is the Greater Philadelphia region's top IT consulting firm specializing in software development, systems integration, mobile application development and training.

Our team includes many of the top software architects in the area, with deep technical expertise, industry knowledge and a genuine passion for software development.

Visit us online at chariotsolutions.com.

CHARIOT SOLUTIONS