DataEngBytes | 2023

Exposing Big Edge Data:

What Big Cloud Providers Don't Want You to Know

Presented by Nathan Glover

INTRODUCTION

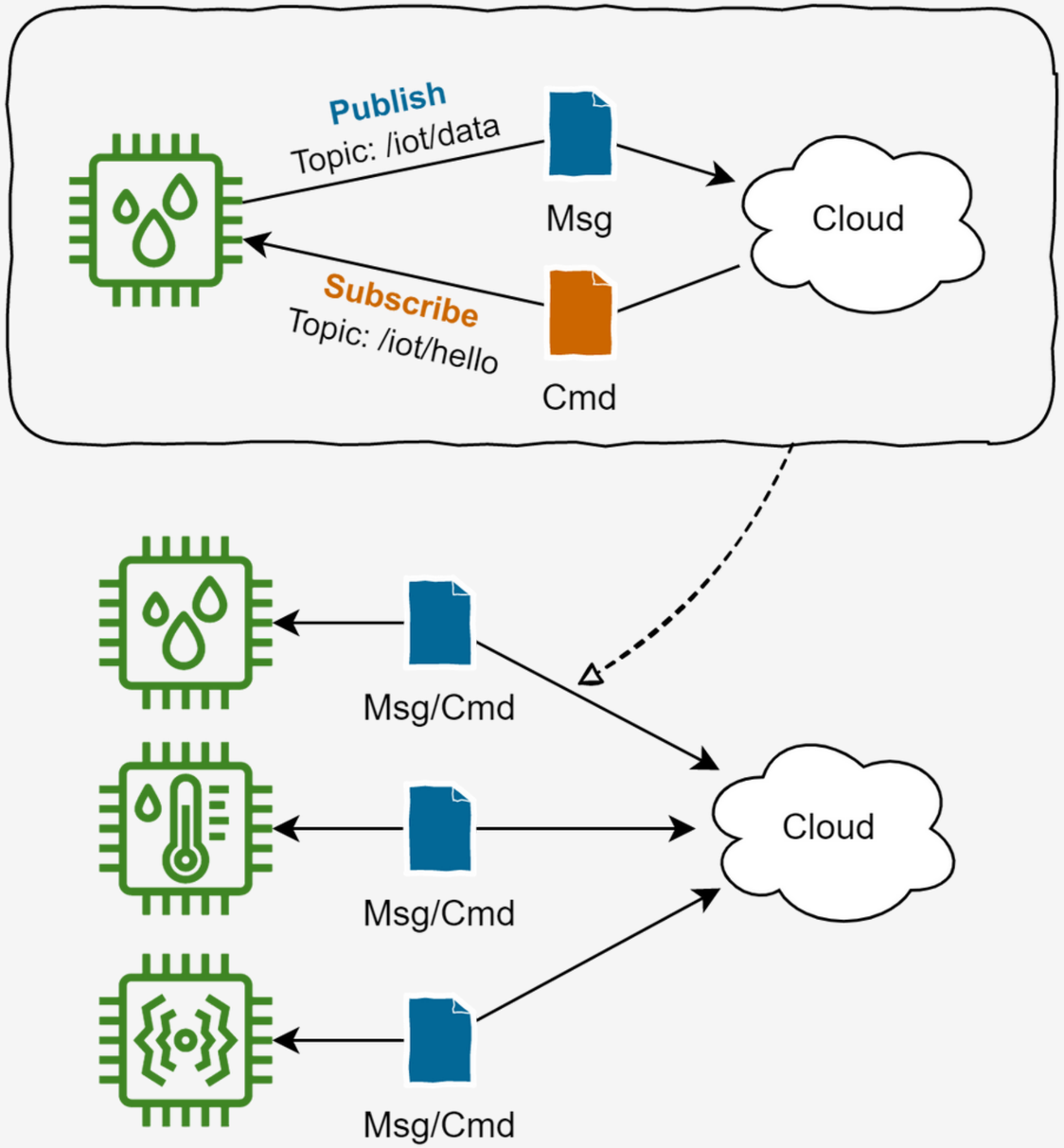
- Senior Platform Engineer @ First Mode
- Aspiring Data Engineer
- AWS IoT Hero
- Lover of cats

WHAT'S THE PLAN

- Popular IoT data ingestion strategies
- Why **direct to S3/blob storage** can be more cost efficient
- How Apache Iceberg can support us
- Simple and robust data pipeline to **Apache Iceberg**
- Access patterns for users and data teams

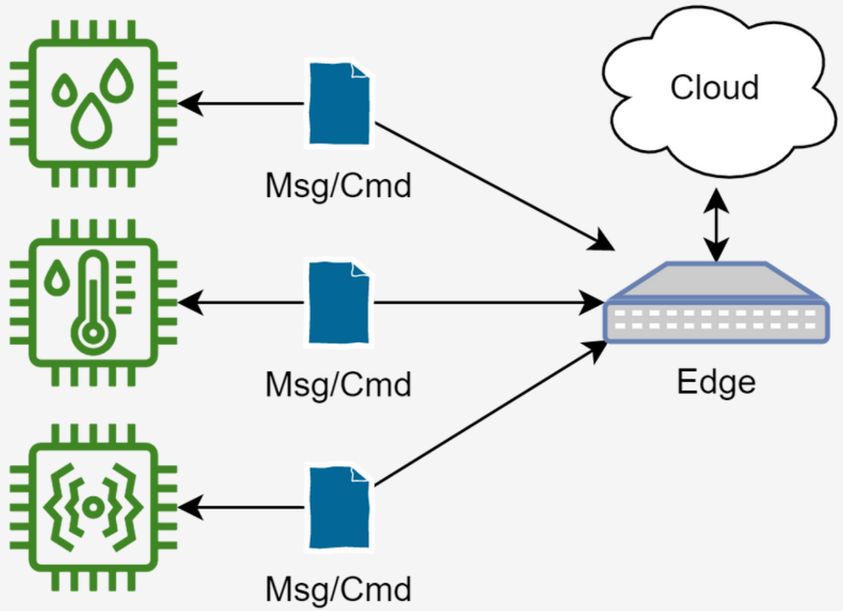
TYPICAL IOT PATTERN(S)

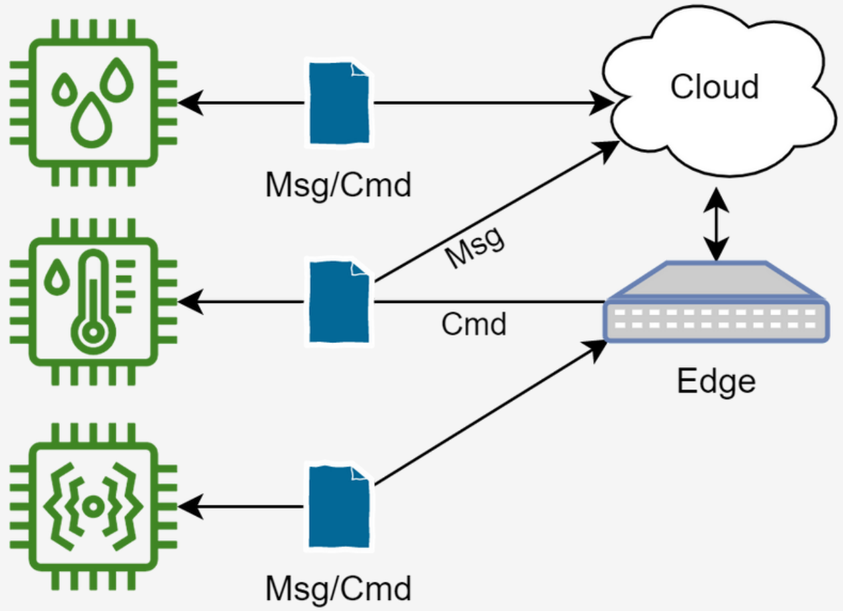
- One or more sensors operating independently of each other
- Communicate over MQTT
- Publish/Subscribe model
- Direct to cloud



TYPICAL IOT PATTERN(S)

- One or more sensors attached to an edge
- Communicate over MQTT, but through an "Edge" device
- Messages are mirrored up to the cloud but actions can be taken at the edge before that

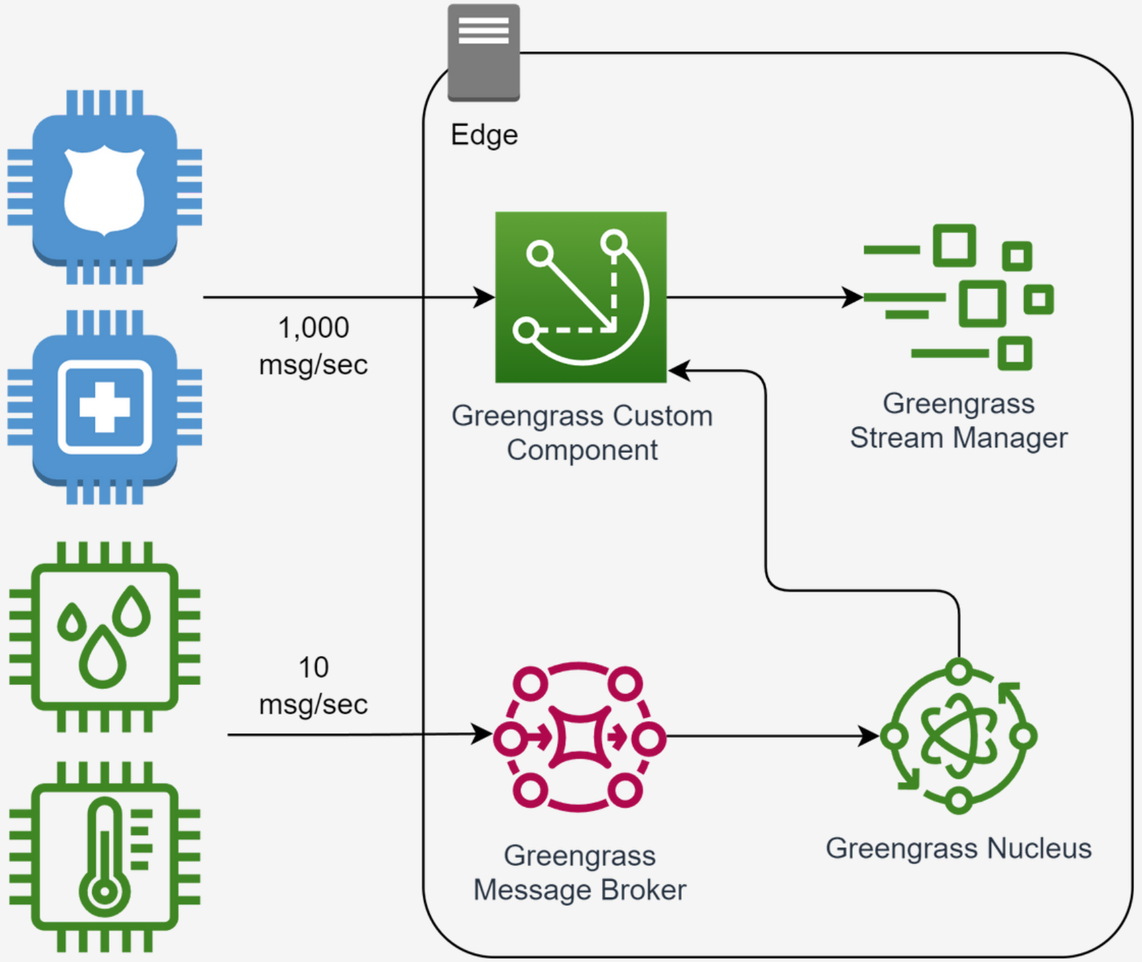




CAVEATS

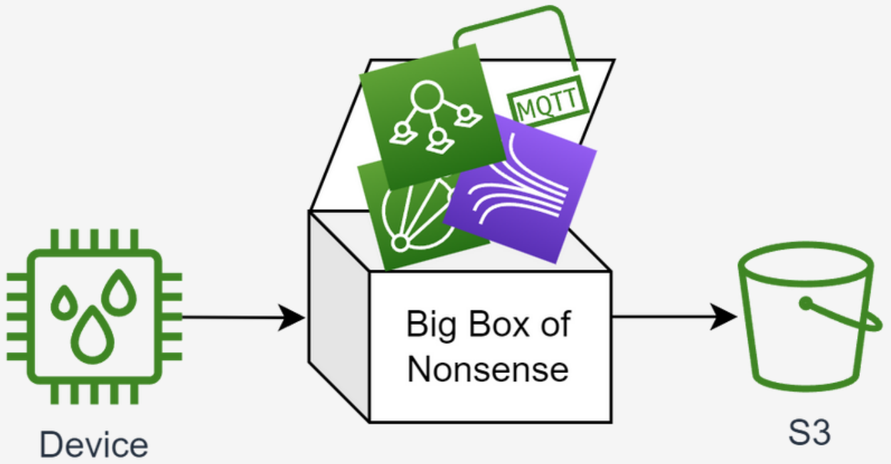
for this talk

- Focus on bulk data ingestion to the cloud
- Ignoring actuation, communication from the cloud to the device
- Non-safety critical
  - This design can complement other, more appropriate designs for those use cases.
- When we talk edge, we're talking about AWS Greengrass
  - Greengrass is the <Managed Everything> for running on the edge.
  - Interoperable with
    - Azure IoT Edge
    - Kubernetes
    - Any compute that can run code to service your sensors



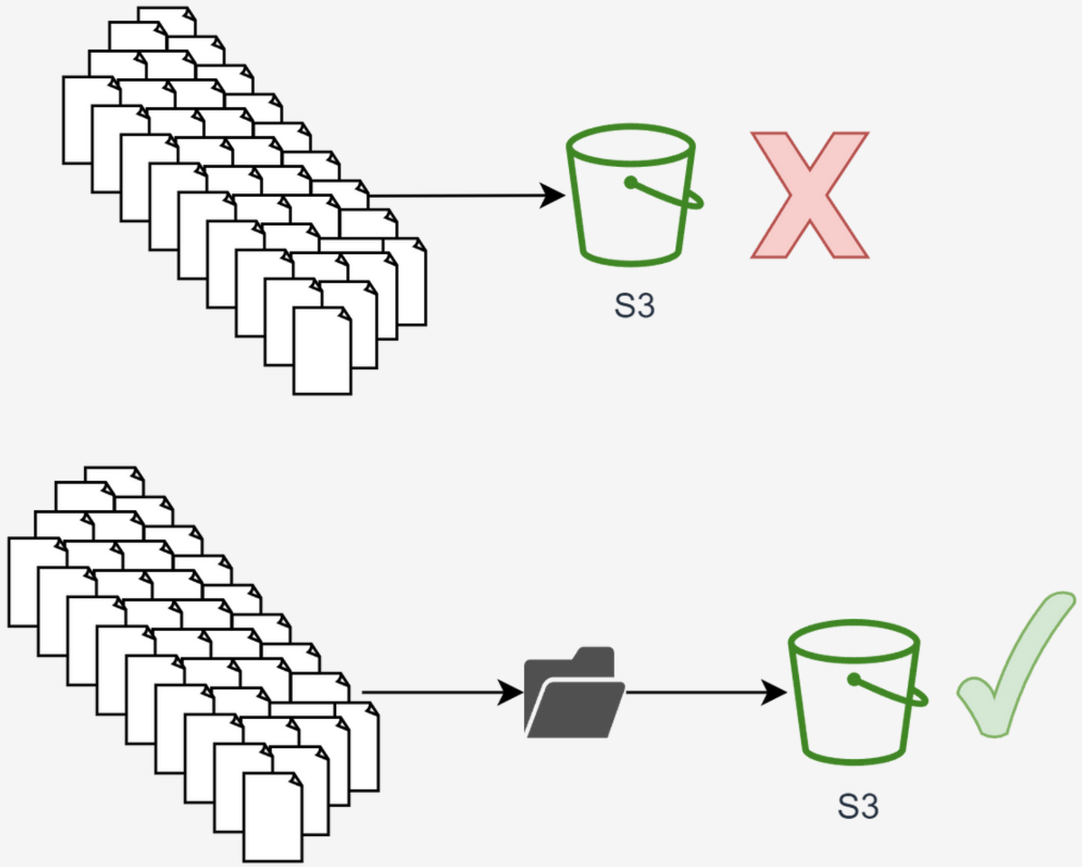
LOWEST COMMON DENOMINATOR

- We noticed that in most situations, data is eventually landed in S3
- Does it make more sense to cut the middleman?
- Do you meaningfully use the features associated with Kinesis or IoT Core?



ISN'T S3 EXPENSIVE?

- Yes, if you use it wrong!

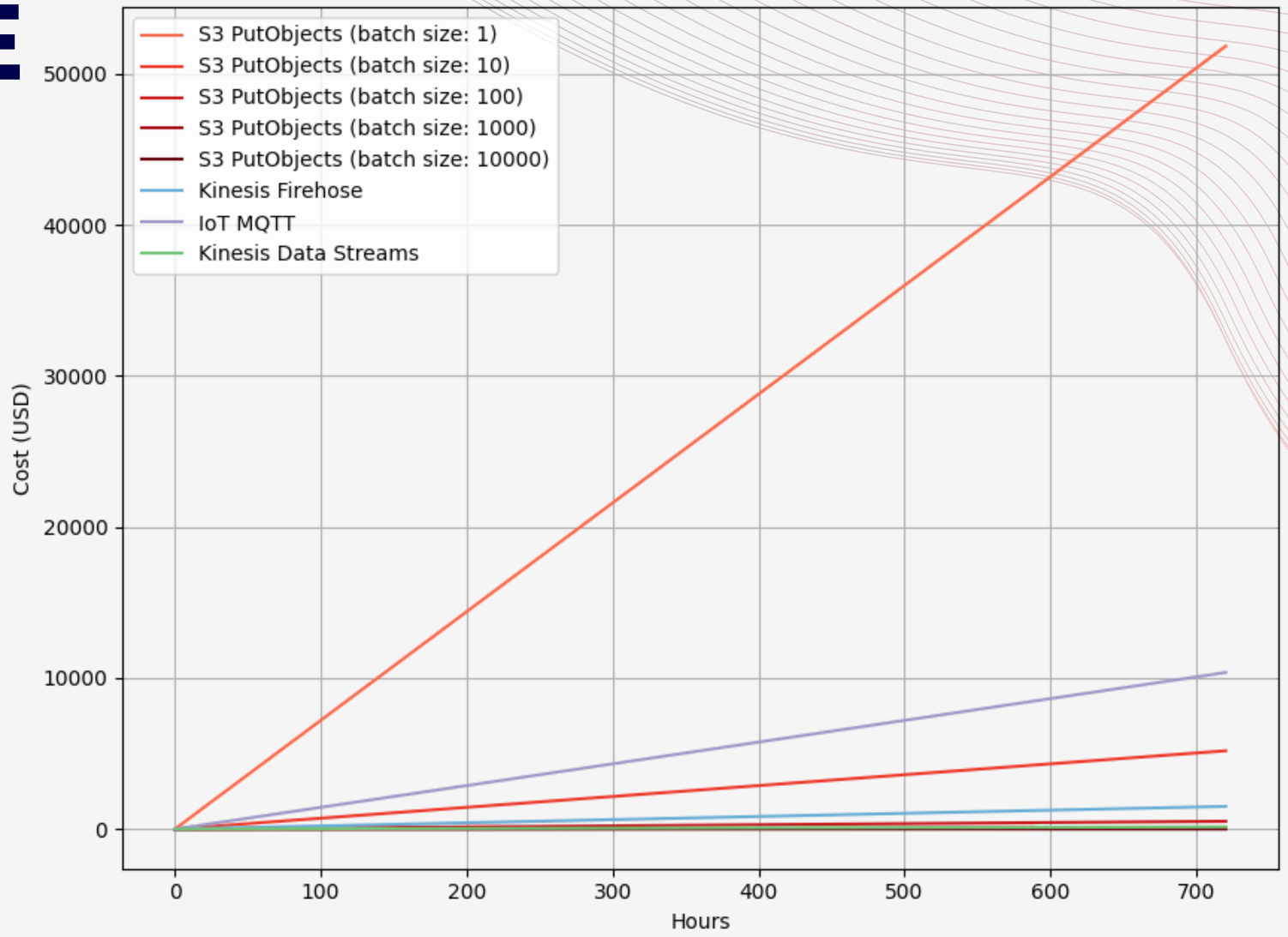


EXPLORE THE COSTS

Data was sent at a rate of 100 messages/sec from 40 devices, a total of 4000 messages/sec.

Each record averages out to 166.25 bytes

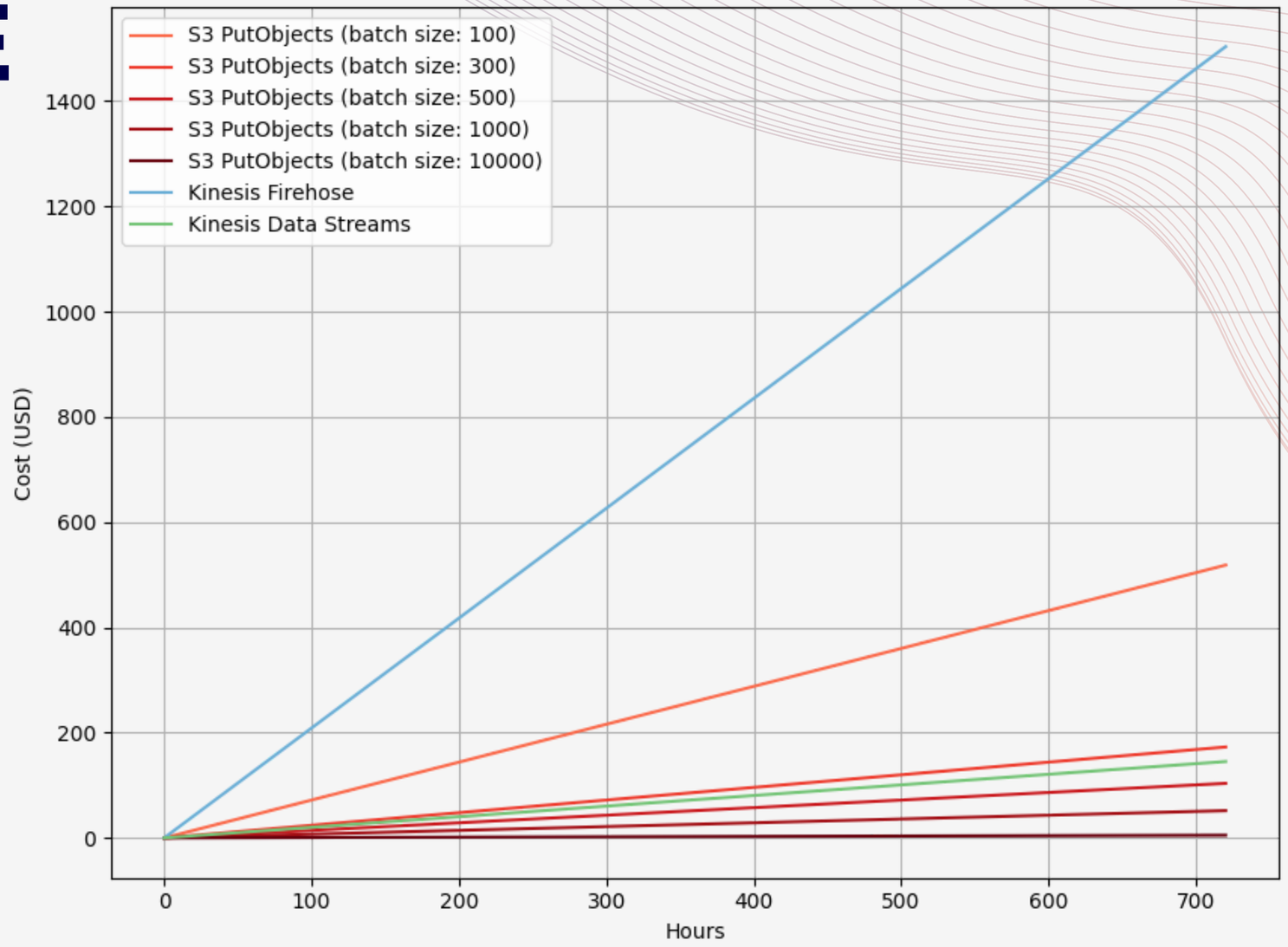
Costs are calculated over up to 720 hours (30 days).



EXPLORE THE COSTS

Kinesis Data Stream is limited to 500 record batches (with the provided clients by Amazon)

Kinesis Firehose has a minimum record size of 5kb.



COST FINDINGS

IOT CORE (MQTT)

- Highest cost is due to it not being appropriate for the use case

KINESIS FIREHOSE

- On the surface, it seems like it would make sense, however, a small pricing quirk makes it much more expensive
- 5k minimum data size, 5kb increments
- right-sizing your data payloads is necessary

KINESIS DATA STREAMS

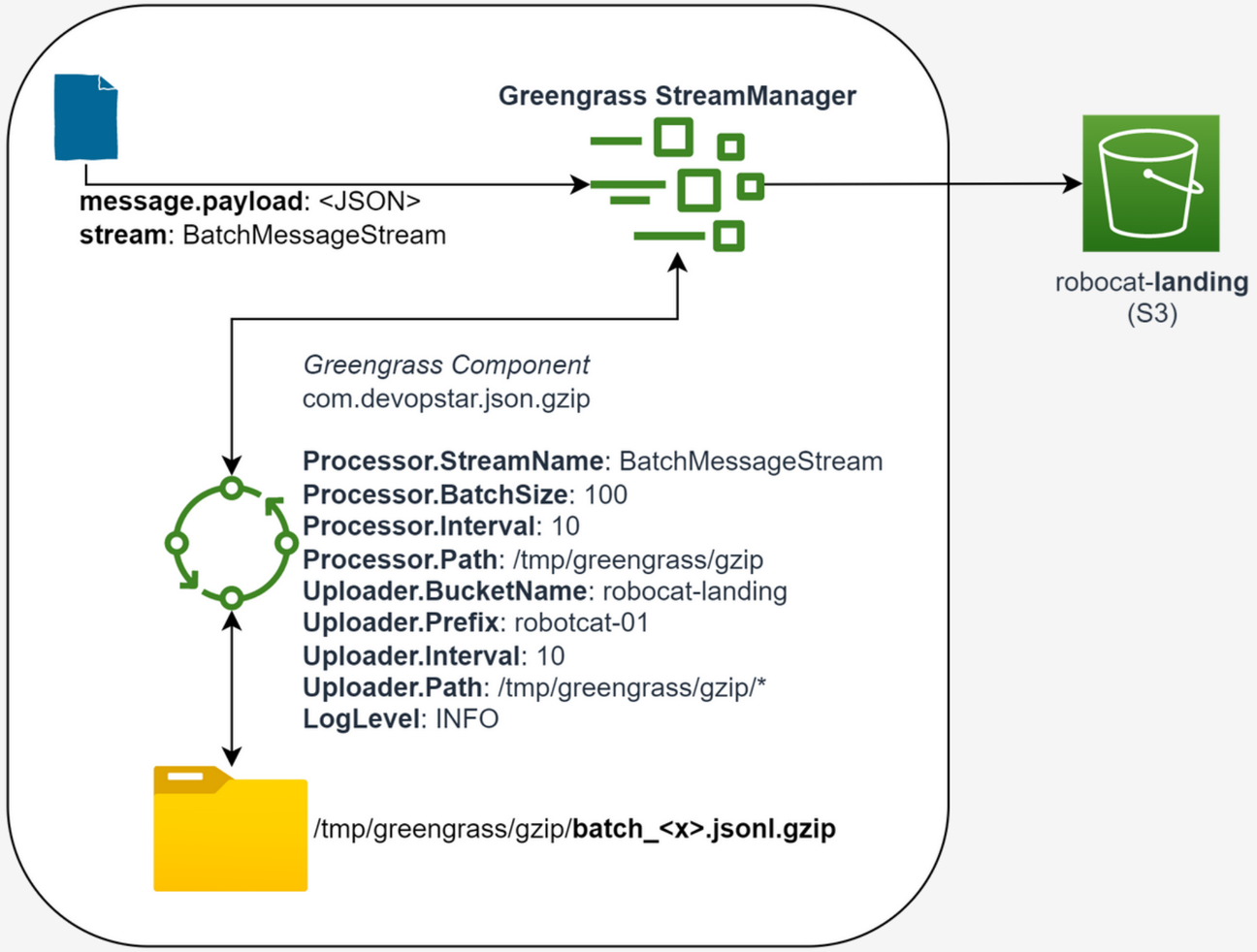
- This is primarily considered to be the wisest solution for this use-case
- Kinesis Producer Library (KPL) is difficult to work with natively and is only available in Java

S3

- Goes from the most costly to the cheapest as the batch size increases
- Technically this could be called "cloud agnostic" given its just blob storage.

PROPOSED DESIGN (OVERVIEW)

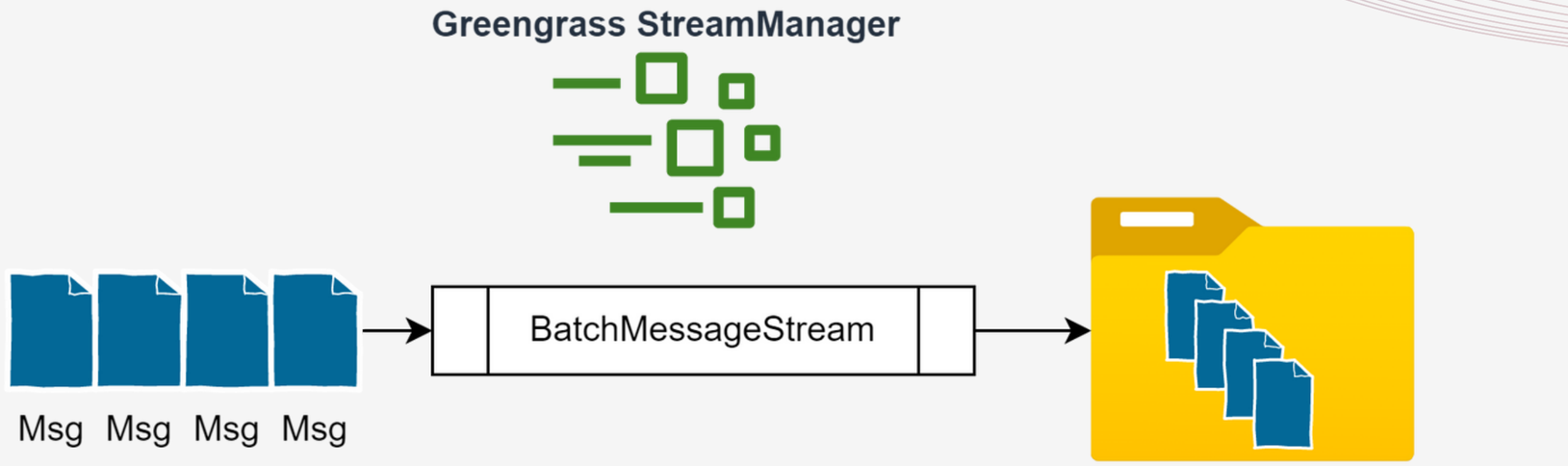
- Take the best parts of Kinesis Firehose and roll it ourselves
- Batch data into a format that can be read by a query engine and compress
- Send compressed data direct to S3
- Use some underappreciated Greengrass components and features.



https://github.com/t04glovern/aws-greengrass-json-gzip

EDGE

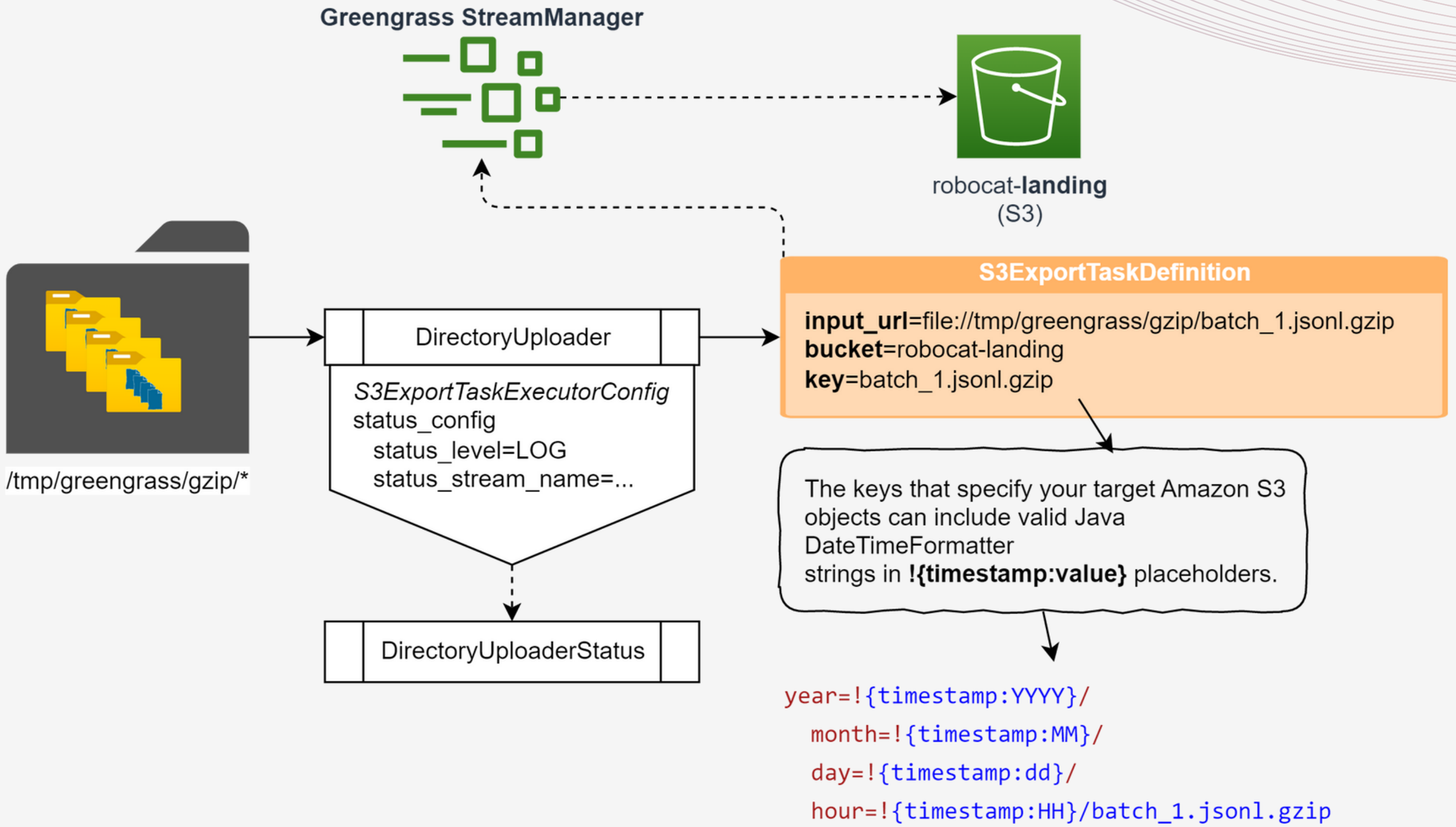
Greengrass Component Breakdown - **Stage 1**




```
stream_manager_client = StreamManagerClient()
stream_manager_client.append_message(
    stream_name="BatchMessageStream",
    data=json.dumps(
        {"message": "Hello World!"}
    ).encode())
```

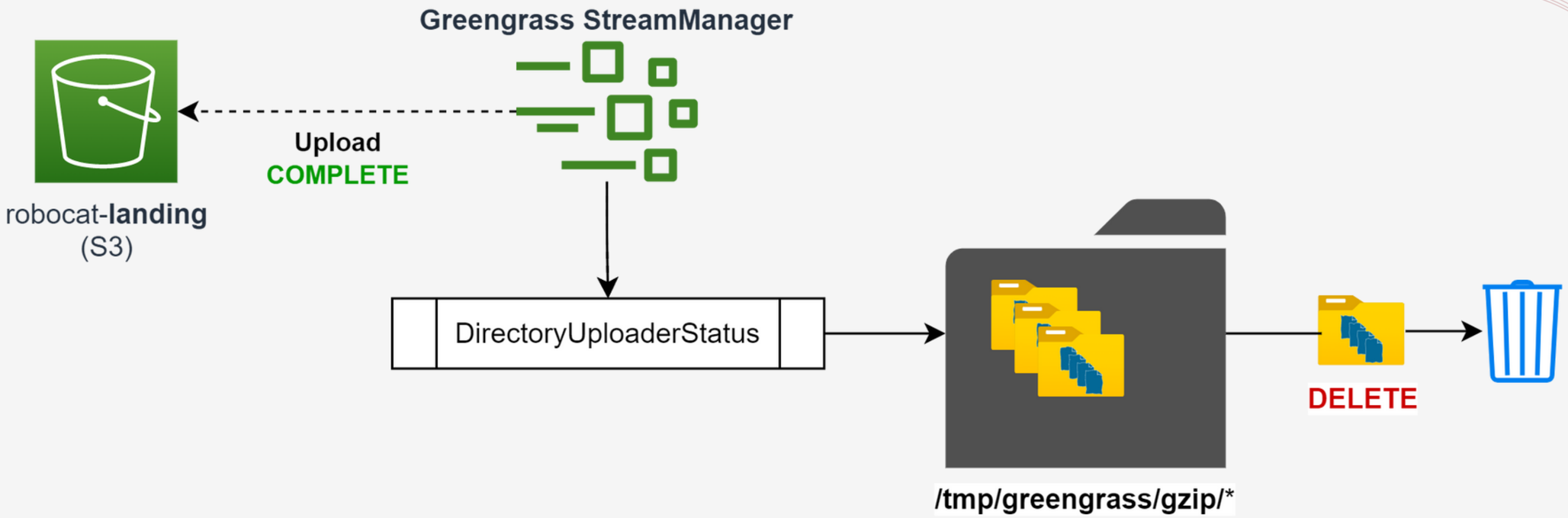
EDGE

Greengrass Component Breakdown - **Stage 2**



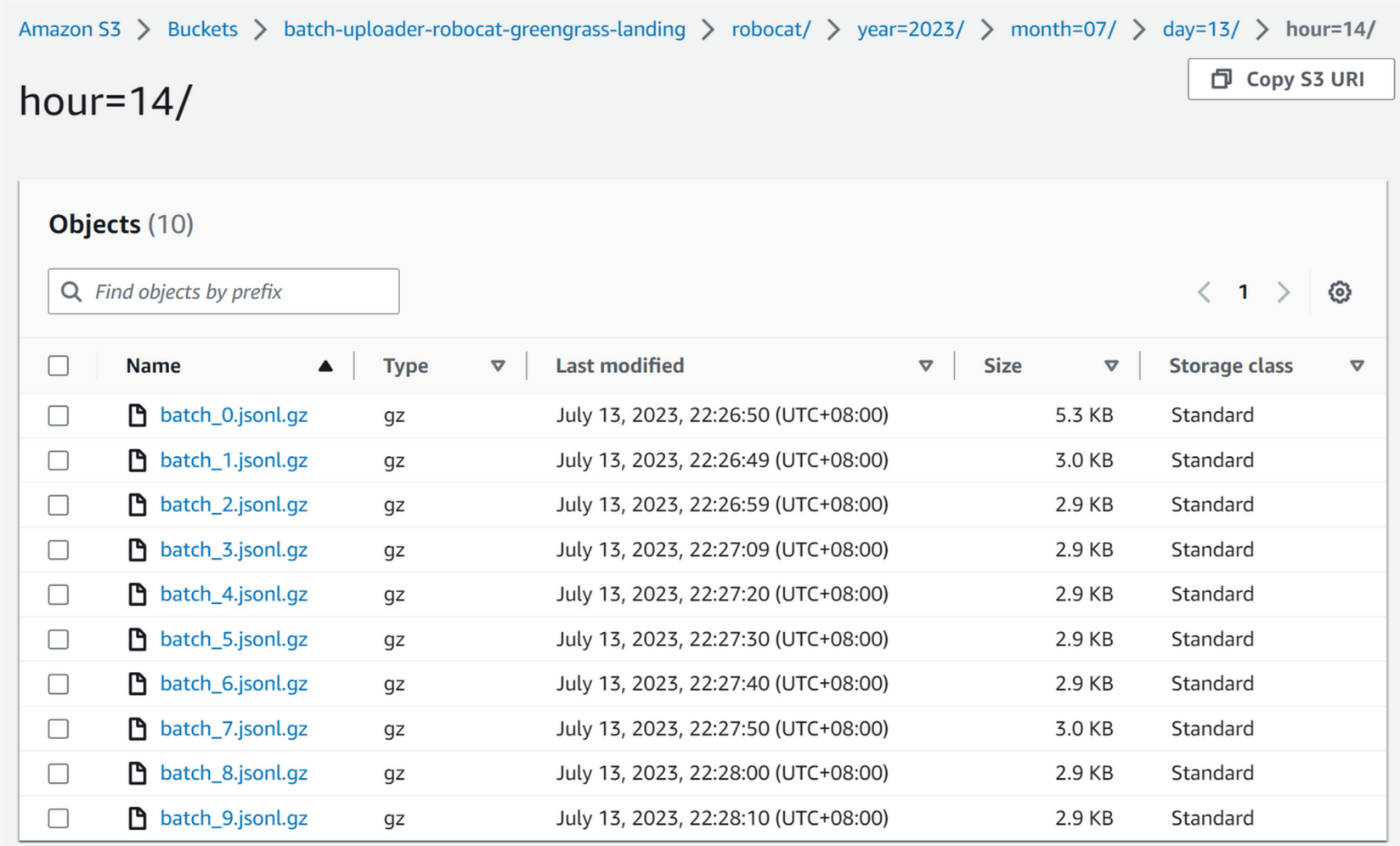
EDGE

Greengrass Component Breakdown - **Stage 3**



QUERY LANDED DATA

- The problem with this
  - Late data, offline processing doesn't write to correct partitions



QUERY LANDED DATA

- Athena Query on top of Landing data

```
CREATE EXTERNAL TABLE IF NOT EXISTS greengrass_data (
    `id` string,
    `timestamp` timestamp,
    `speed` int,
    `temperature` float,
    `location` struct < lat: float, lng: float >
)
PARTITIONED BY ( year int, month int, day int, hour int )
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
WITH SERDEPROPERTIES ( "timestamp.formats"="yyyy-MM-dd'T'HH:mm:ss.SSSSSSZZ" )
LOCATION 's3://batch-uploader-robocat-greengrass-landing/robocat/'
TBLPROPERTIES (
    "projection.enabled" = "true",
    "projection.year.type" = "integer",
    "projection.year.range" = "2023,2033",
    "projection.month.type" = "integer",
    "projection.month.range" = "1,12",
    "projection.month.digits" = "2",
    "projection.day.type" = "integer",
    "projection.day.range" = "1,31",
    "projection.day.digits" = "2",
    "projection.hour.type" = "integer",
    "projection.hour.range" = "0,23",
    "projection.hour.digits" = "2",
    "storage.location.template" = "s3://batch-uploader-robocat-greengrass-
landing/robocat/year=${year}/month=${month}/day=${day}/hour=${hour}"
);
```

QUERY LANDED DATA

- Demonstrate the basic functionality

```
SELECT *
FROM "default"."greengrass_data"
WHERE year = 2023
    AND month = 7
    AND day = 12
    AND hour = 14
```

Query results | Query stats

Completed — Time in queue: 98 ms — Run time: 957 ms — Data scanned: 24.65 KB

Results (845)

Copy | Download results

Search rows

1 ...

#	id	timestamp	speed	temperature	location	year	month	day	hour
1	1	2023-07-12 14:13:20.563	52	20.8	{lat=-31.969883, lng=115.878716}	2023	7	12	14
2	1	2023-07-12 14:13:20.670	52	20.41	{lat=-31.969313, lng=115.8787}	2023	7	12	14
3	1	2023-07-12 14:13:20.775	52	20.66	{lat=-31.969738, lng=115.87843}	2023	7	12	14
4	1	2023-07-12 14:13:20.879	51	20.43	{lat=-31.970194, lng=115.878975}	2023	7	12	14
5	1	2023-07-12 14:13:20.984	51	20.28	{lat=-31.970276, lng=115.878136}	2023	7	12	14
6	1	2023-07-12 14:13:21.089	51	20.3	{lat=-31.969902, lng=115.87753}	2023	7	12	14
7	1	2023-07-12 14:13:21.194	51	20.11	{lat=-31.970348, lng=115.878296}	2023	7	12	14
8	1	2023-07-12 14:13:21.298	51	20.14	{lat=-31.970379, lng=115.87902}	2023	7	12	14
9	1	2023-07-12 14:13:21.402	51	19.93	{lat=-31.969402, lng=115.87956}	2023	7	12	14

QUERY LANDED DATA (BONUS)

- Schema? never heard of it

```
CREATE EXTERNAL TABLE IF NOT EXISTS greengrass_json_data (
  jsonstring string
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
  "input.regex" = "^(.*)$",
  "projection.enabled" = "true",
  "projection.year.type" = "integer",
  "projection.year.range" = "2023,2033",
  "projection.month.type" = "integer",
  "projection.month.range" = "1,12",
  "projection.month.digits" = "2",
  "projection.day.type" = "integer",
  "projection.day.range" = "1,31",
  "projection.day.digits" = "2",
  "projection.hour.type" = "integer",
  "projection.hour.range" = "0,23",
  "projection.hour.digits" = "2",
  "storage.location.template"="s3://batch-uploader-robocat-greengrass-
landing/robocat/year=${year}/month=${month}/day=${day}/hour=${hour}"
) LOCATION 's3://batch-uploader-robocat-greengrass-landing/robocat/';
```

QUERY LANDED DATA (BONUS)

- Schema? never heard of it

```
SELECT * FROM "default"."greengrass_json_data" limit 10
```

Completed | Time in queue: 102 ms | Run time: 600 ms | Data scanned: 8.75 KB

Results (10)

Copy | Download results

Search rows

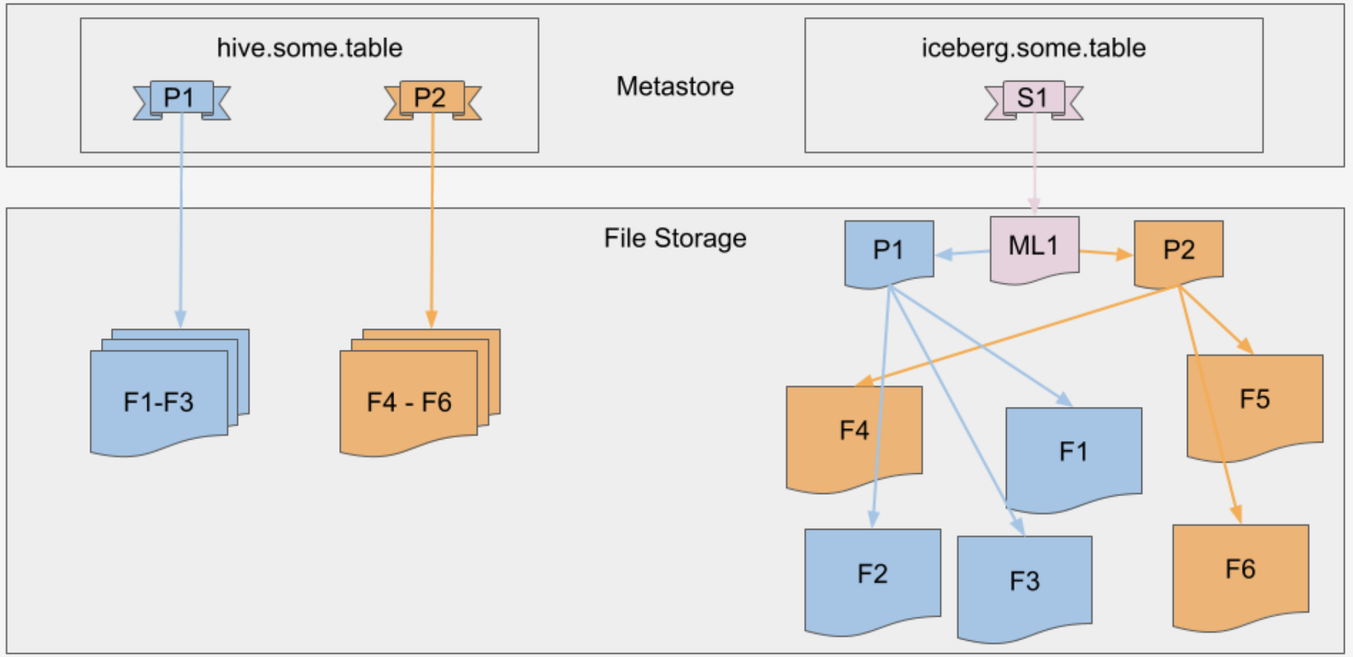
1

#	jsonstring
1	{"id":"1","timestamp":"2023-07-13T14:27:43.727113+00:00","speed":47.72,"temperature":9.28,"location":{"lat":-31.959397733165545,"lng":115.89989853611982}}
2	{"id":"1","timestamp":"2023-07-13T14:27:43.831589+00:00","speed":47.84,"temperature":9.4,"location":{"lat":-31.95844675372861,"lng":115.89972415086329}}
3	{"id":"1","timestamp":"2023-07-13T14:27:43.935858+00:00","speed":48.16,"temperature":9.78,"location":{"lat":-31.958945186576024,"lng":115.90072390858766}}
4	{"id":"1","timestamp":"2023-07-13T14:27:44.040010+00:00","speed":47.98,"temperature":9.72,"location":{"lat":-31.95817274560059,"lng":115.89987579679638}}
5	{"id":"1","timestamp":"2023-07-13T14:27:44.144238+00:00","speed":48.0,"temperature":9.74,"location":{"lat":-31.957922961466995,"lng":115.89897024319039}}
6	{"id":"1","timestamp":"2023-07-13T14:27:44.248718+00:00","speed":48.24,"temperature":9.5,"location":{"lat":-31.95882830183242,"lng":115.89812315023816}}
7	{"id":"1","timestamp":"2023-07-13T14:27:44.353758+00:00","speed":48.11,"temperature":9.3,"location":{"lat":-31.958415349710833,"lng":115.89858057726154}}
8	{"id":"1","timestamp":"2023-07-13T14:27:44.462630+00:00","speed":47.85,"temperature":9.32,"location":{"lat":-31.958034706122458,"lng":115.89956419571149}}
9	{"id":"1","timestamp":"2023-07-13T14:27:44.570846+00:00","speed":48.13,"temperature":9.3,"location":{"lat":-31.957290339030926,"lng":115.90030159135453}}
10	{"id":"1","timestamp":"2023-07-13T14:27:44.683935+00:00","speed":48.15,"temperature":9.44,"location":{"lat":-31.95704576863196,"lng":115.9011041907042}}

APACHE ICEBERG

Let's solve that pesky landing partition problem

- A next-generation table format for big data analytics.
- Hidden Partitioning: Efficiently manages large datasets.
- Schema & Partition Evolution
- Fast Plan & Execution
  - Metadata allows Iceberg to know exactly what files are needed

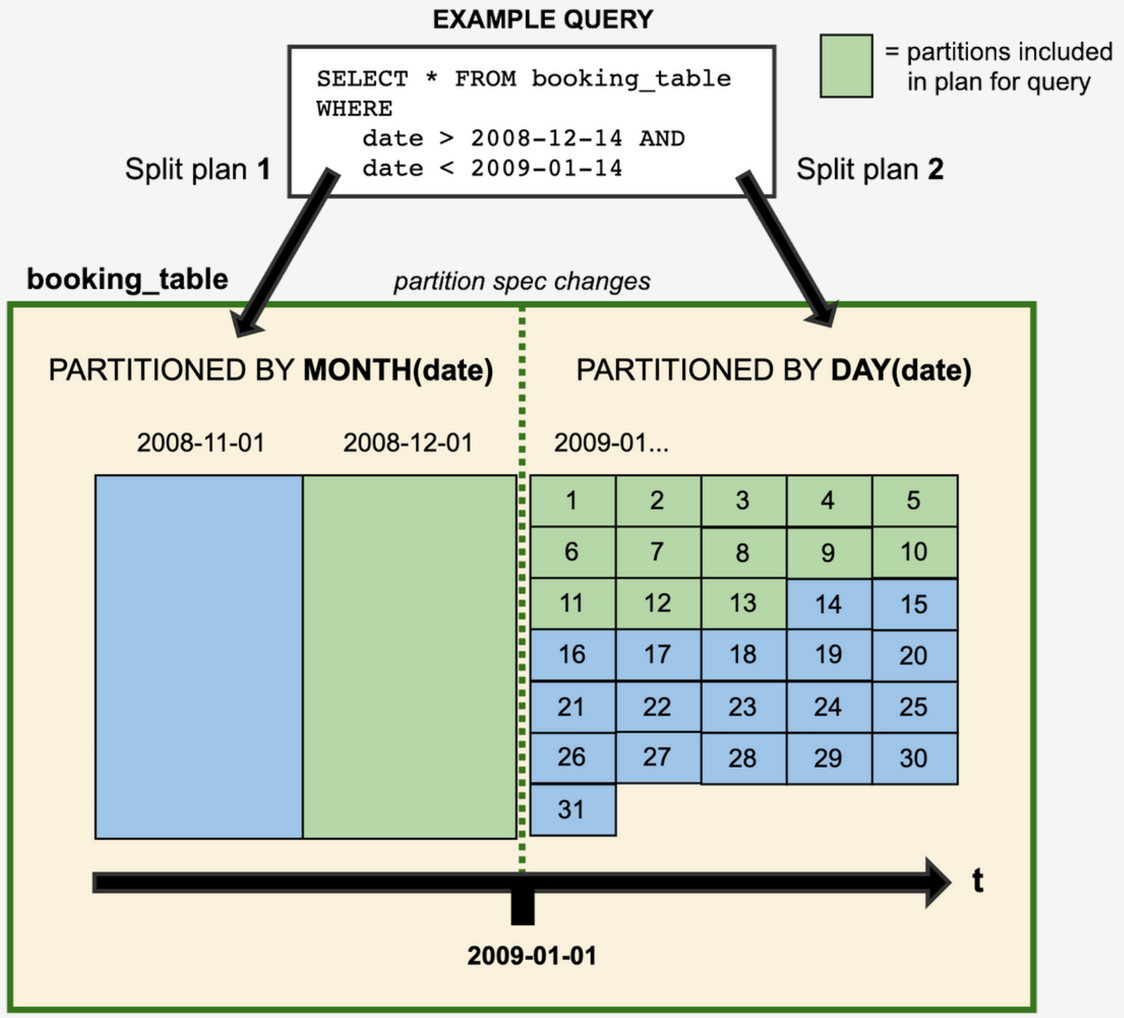


https://www.starburst.io/blog/trino-on-ice-ii-in-place-table-evolution-and-cloud-compatibility-with-iceberg/

APACHE ICEBERG

Partition Evolution

- If you change the partition spec, old data under this spec is unchanged
- "Hidden" partitioning means you don't need to write a query for a given partition
  - Just write a query, and iceberg does its thing!



https://iceberg.apache.org/docs/latest/evolution/#partition-evolution

APACHE ICEBERG

Schema Evolution

- Similar benefits as Partition Evolution
- Iceberg can handle schema changes
  - Adding a column back won't result in "zombie" data coming back from the dead.

- **Add** – add a new column to the table or to a nested struct
- **Drop** – remove an existing column from the table or a nested struct
- **Rename** – rename an existing column or field in a nested struct
- **Update** – widen the type of a column, struct field, map key, map value, or list element
- **Reorder** – change the order of columns or fields in a nested struct

https://iceberg.apache.org/docs/latest/evolution/#schema-evolution

APACHE ICEBERG

Snapshots and Timetravel

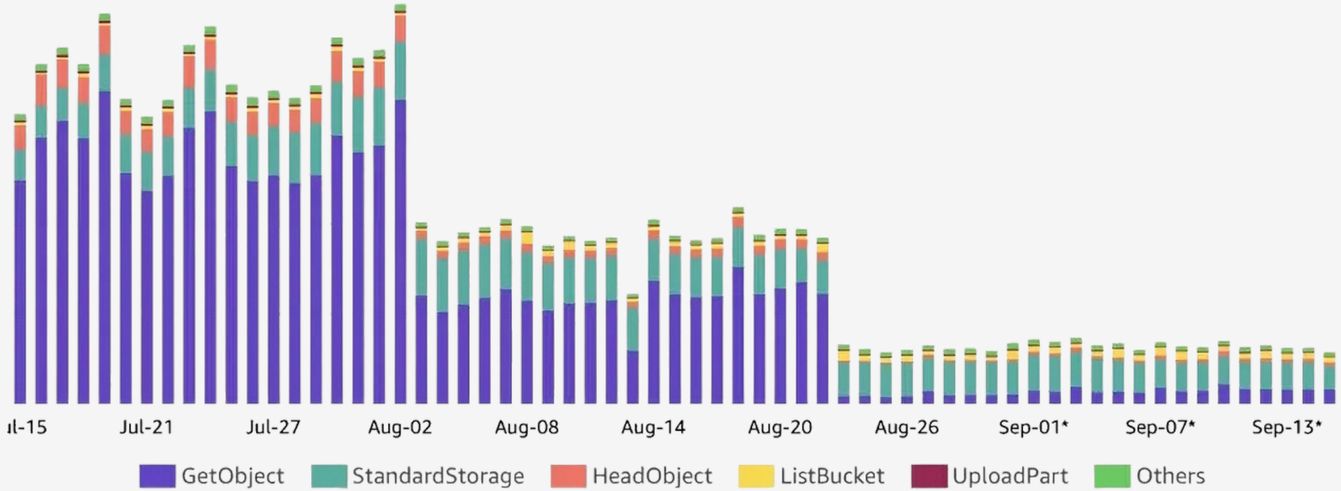
- Each write to an Iceberg table creates a snapshot (version of a table)
- Snapshots require metadata to be stored
  - can balloon out to more than your actual storage without maintenance.
- We'll talk about Maintenance later!

```
SELECT count(*) FROM nyc.taxis
2,853,020

SELECT count(*) FROM nyc.taxis
FOR VERSION AS OF 21884653078355854 43
2,798,371

SELECT count(*) FROM nyc.taxis
FOR TIMESTAMP AS OF TIMESTAMP '2022-01-01 00:00:00.000000 Z'
2,798,371
```

APACHE ICEBERG

Cost Vs. Hive

- **Head/GetObject** requests comprise most (90%) of the cost.
- Iceberg can be configured to merge data to target a file size
  - write.target-file-size-bytes



https://medium.com/insiderengineering/apache-iceberg-reduced-our-amazon-s3-cost-by-90-997cde5ce931

APACHE ICEBERG

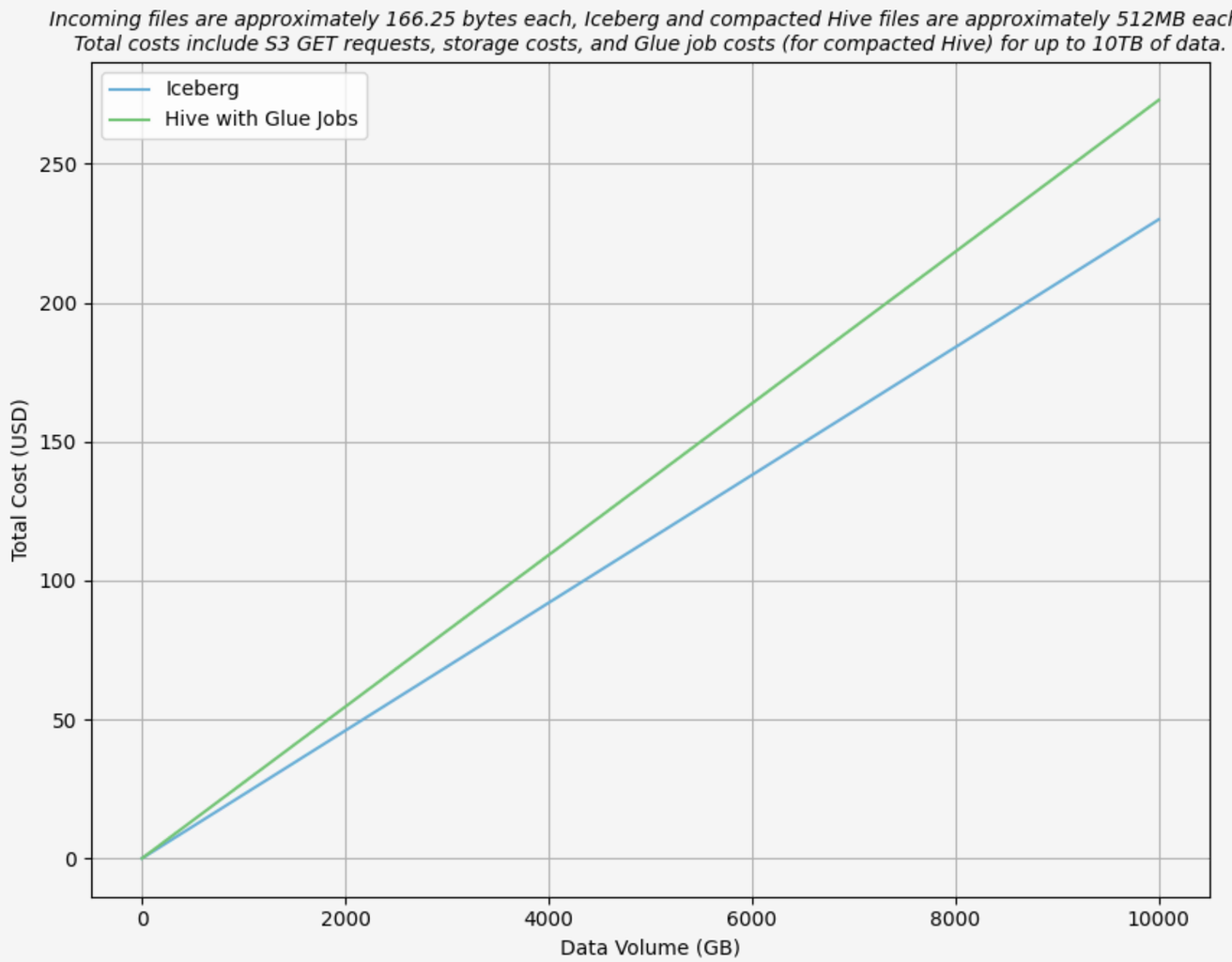
Cost Vs. Hive cont.

- Hive without compaction is about twice the cost in the scenario I've cooked up here.
  - 16,625 bytes per file (100 records)
- You would never want to do this.

APACHE ICEBERG

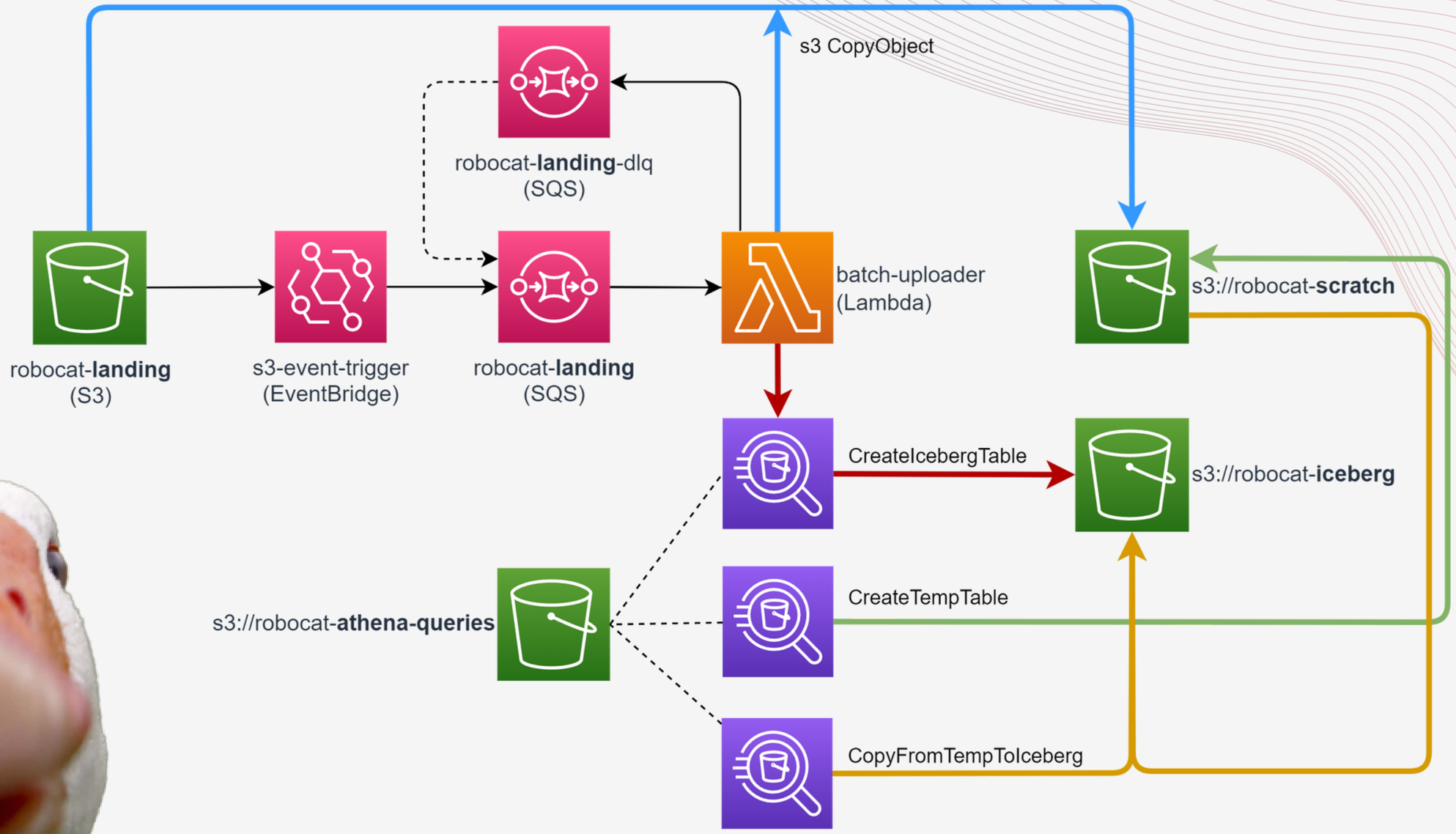
Cost Vs. Hive cont.

- Using Glue jobs and compacting hive partitions can significantly improve costs - but....
- Lots of work to setup
  - Either compact after data lands and the next partition begins (deal with breaking queries in flight on old data)
  - Compact as it lands and, possibly have a significant delay on data



CLOUD PROCESSING

Overview

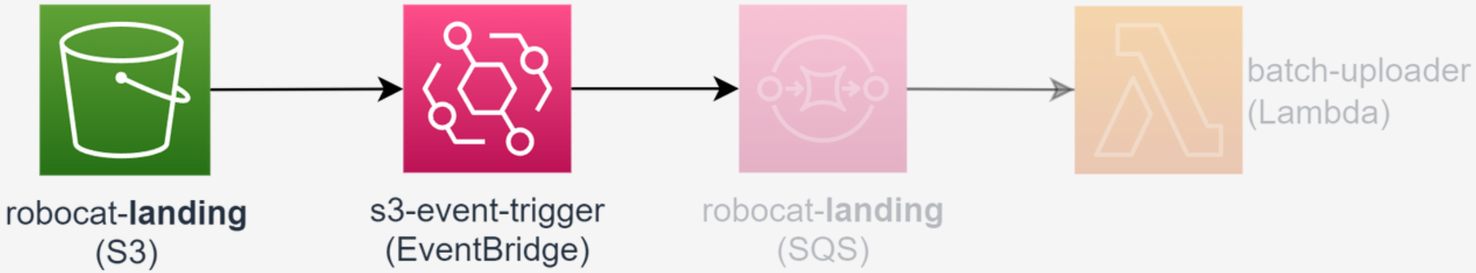


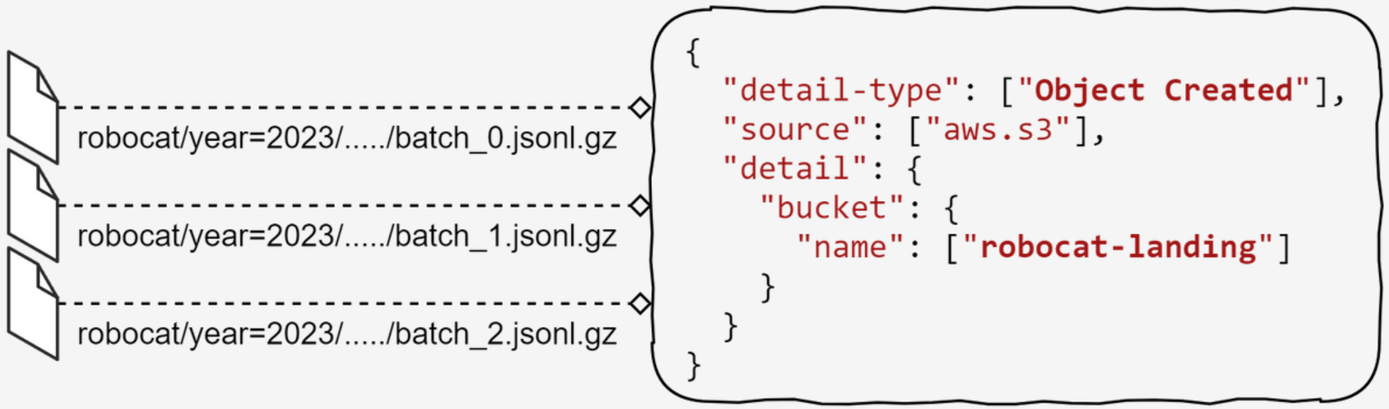
Don't be intimidated!

LOOK AT THIS GOOSE AND ILL EXPLAIN

CLOUD PROCESSING

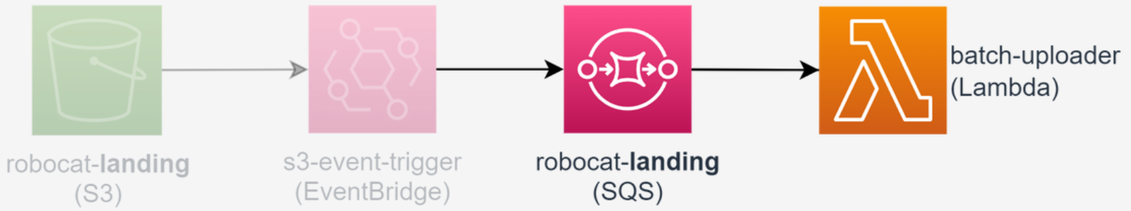
Data Landing Events - **Stage 1**





CLOUD PROCESSING

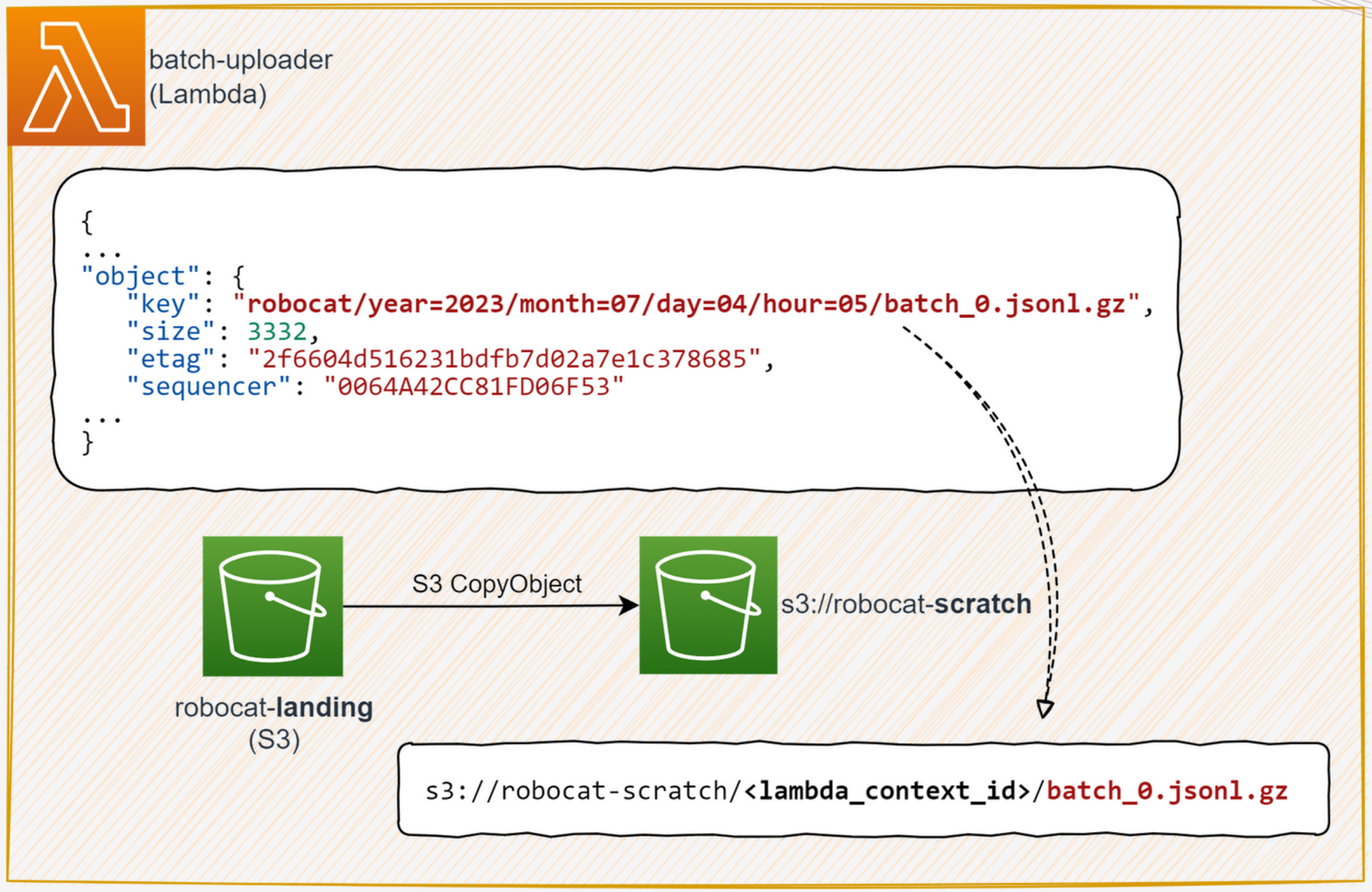
Data Landing Event SQS - **Stage 2**




```
{
    "version": "0",
    "id": "ec9a27c5-db62-373f-09d0-516a37027c6f",
    "detail-type": "Object Created",
    "source": "aws.s3",
    "account": "536829251200",
    "time": "2023-07-04T14:29:28Z",
    "region": "ap-southeast-2",
    "resources": [
        "arn:aws:s3:::robocat-landing"
    ],
    "detail": {
        "version": "0",
        "bucket": {
            "name": "robocat-landing"
        },
        "object": {
            "key": "robocat/year=2023/month=07/day=04/hour=05/batch_0.jsonl.gz",
            "size": 3332,
            "etag": "2f6604d516231bdfb7d02a7e1c378685",
            "sequencer": "0064A42CC81FD06F53"
        },
        "request-id": "HM0KMYCWTE3NRNE5",
        "requester": "536829251200",
        "source-ip-address": "33.22.111.00",
        "reason": "PutObject"
    }
}
```

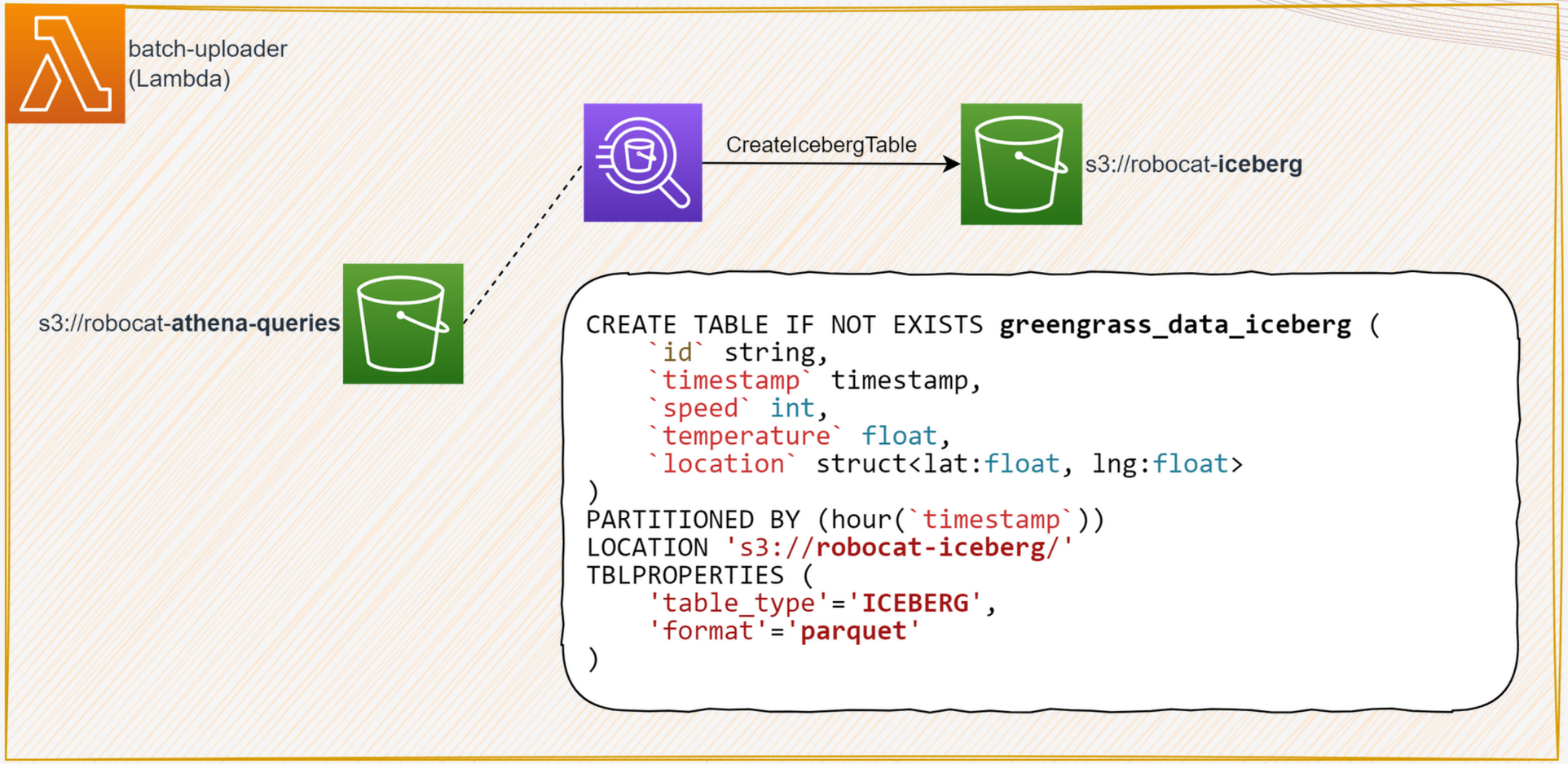
CLOUD PROCESSING

Copy Batch to Scratch - **Stage 3**



CLOUD PROCESSING

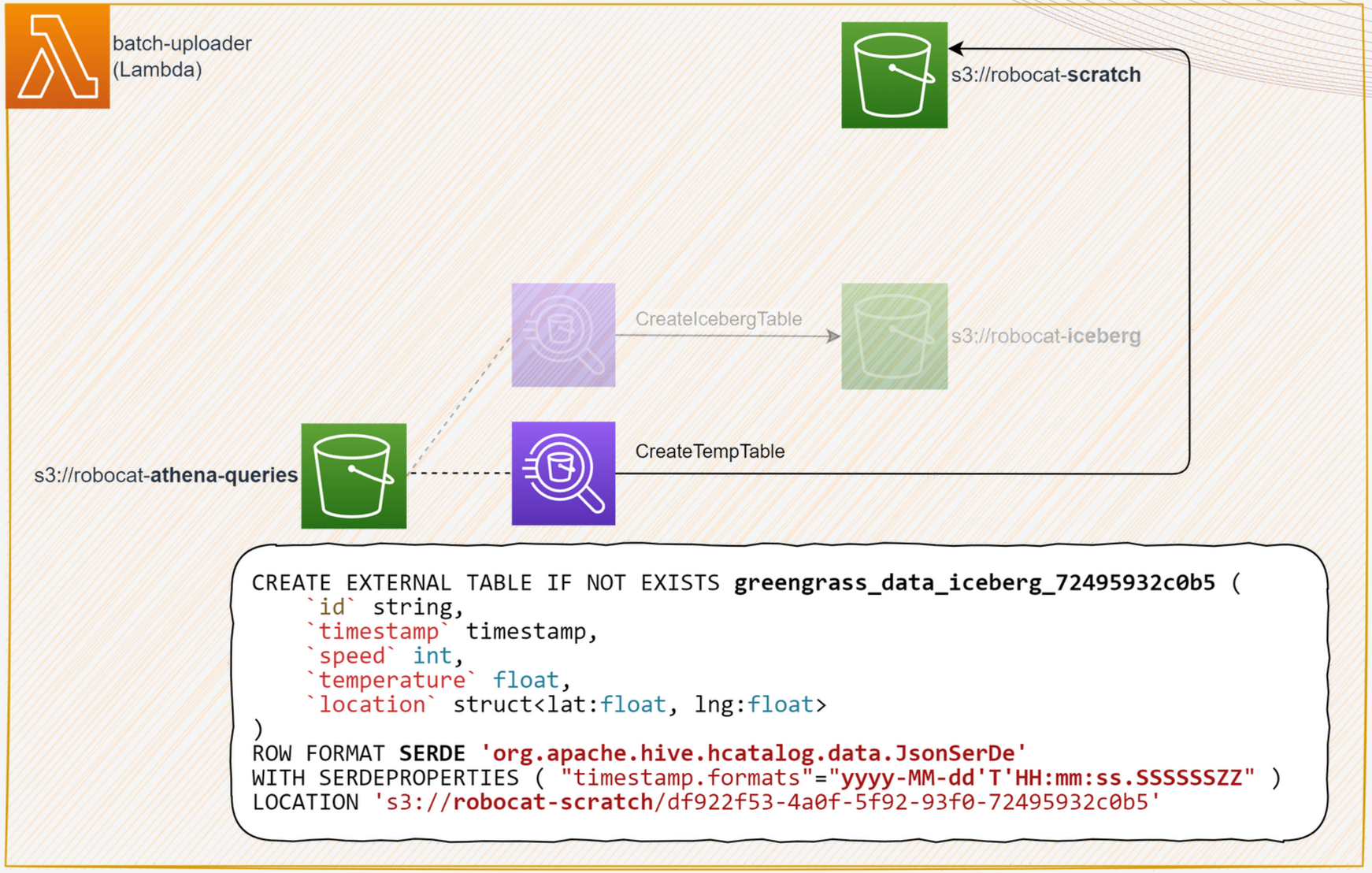
Create Iceberg - **Stage 4**




```
CREATE TABLE IF NOT EXISTS greengrass_data_iceberg (
    `id` string,
    `timestamp` timestamp,
    `speed` int,
    `temperature` float,
    `location` struct<lat:float, lng:float>
)
PARTITIONED BY (hour(`timestamp`))
LOCATION 's3://robocat-iceberg/'
TBLPROPERTIES (
    'table_type'='ICEBERG',
    'format'='parquet'
)
```

CLOUD PROCESSING

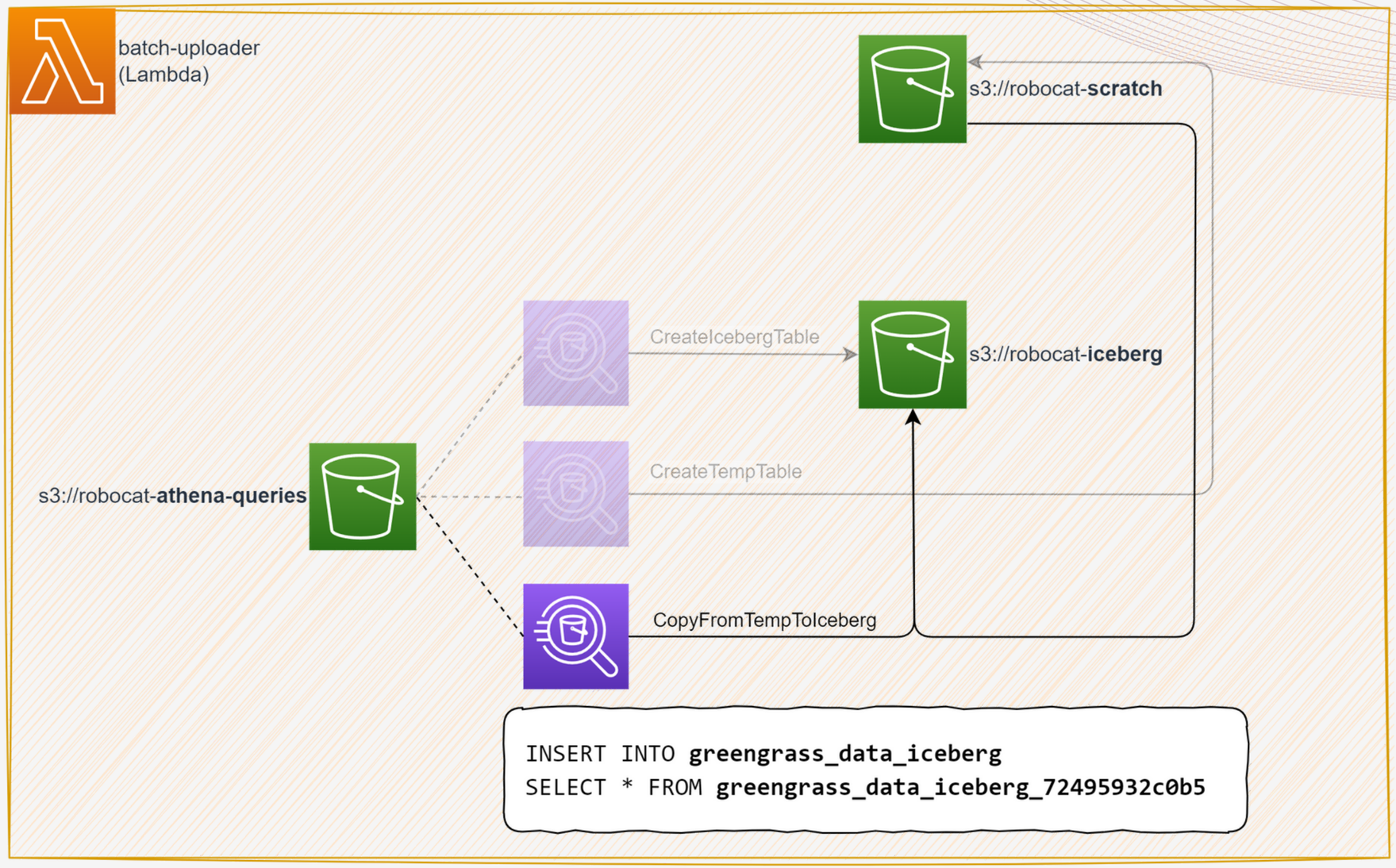
Create Temporary Table - **Stage 4**




```
CREATE EXTERNAL TABLE IF NOT EXISTS greengrass_data_iceberg_72495932c0b5 (
    `id` string,
    `timestamp` timestamp,
    `speed` int,
    `temperature` float,
    `location` struct<lat:float, lng:float>
)
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
WITH SERDEPROPERTIES ( "timestamp.formats"="yyyy-MM-dd'T'HH:mm:ss.SSSSSSZZ" )
LOCATION 's3://robocat-scratch/df922f53-4a0f-5f92-93f0-72495932c0b5'
```

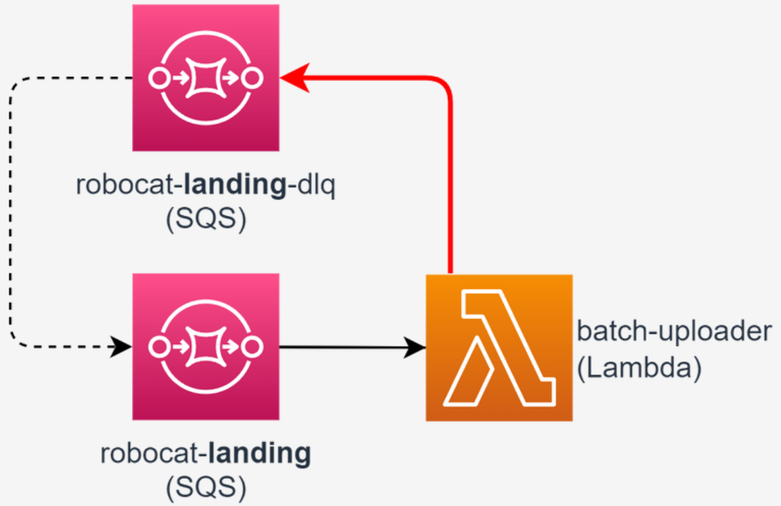
CLOUD PROCESSING

Insert into Iceberg from Temporary Table - **Stage 4**



CLOUD PROCESSING

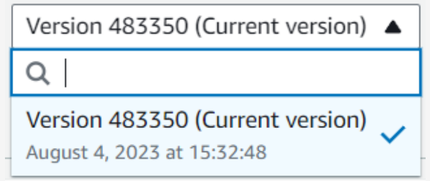
- What about failures?




```python
def start_message_move_task():
    sqs = boto3.client('sqs')

    source_arn = 'arn:xxxx:robocat-landing-dlq'
    dest_arn = 'arn:xxxx:robocat-landing'
    try:
        sqs.start_message_move_task(
            SourceArn=source_arn,
            DestinationArn=dest_arn
        )
    except ClientError as e:
        print(e)
```

ICEBERG HITS DIFFERENT

TABLE MAINTENANCE



- Vacuum and Optimize
- Frequent writes means a lot of snapshots
  - **glue.skip-archive** disables this

SCHEMA CHANGES

- There is currently no structured tooling for handling schema changes. It's all raw SQL, or Iceberg SDK calls yourself.

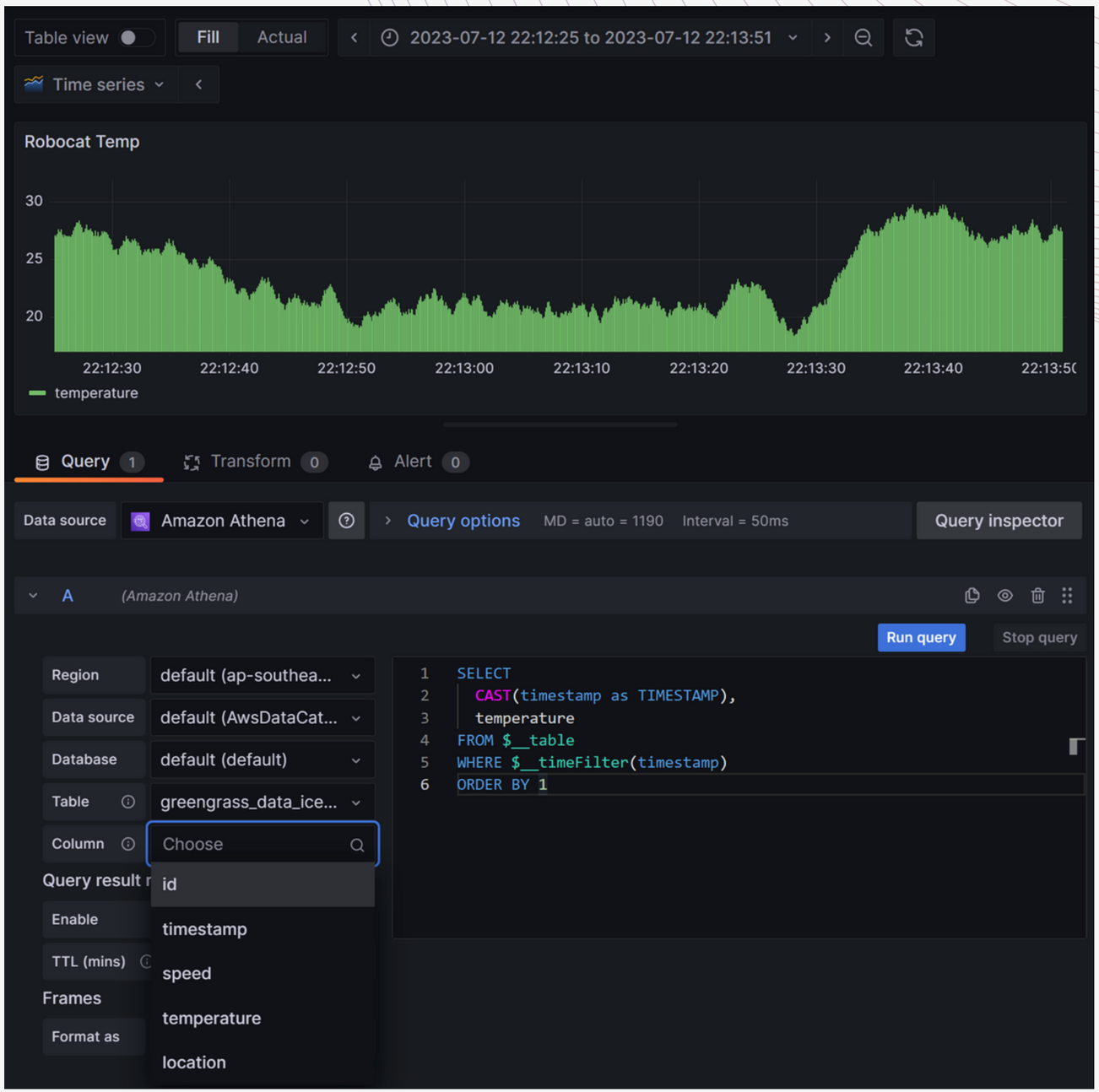
"ATHENA" ICERBERG

- The AWS version has some odd behaviour
- Data pathing cannot contain `=` characters
- Snapshot property reported by Glue is in **milliseconds** but is labelled **seconds** in API responses
  - AWS told us they would change the documentation to fix this??
- Dropped columns will still show up in Glue and Athena
  - AWS support say this is by design, to support Lake Formation

```
# Table properties:
# key   value
vacuum_max_snapshot_age_seconds 86400000
write_compression       gzip
format  parquet
vacuum_min_snapshots_to_keep    5

# Iceberg storage table properties:
# key   value
history.expire.max-snapshot-age-ms      86400000
write.delete.parquet.compression-codec  gzip
```

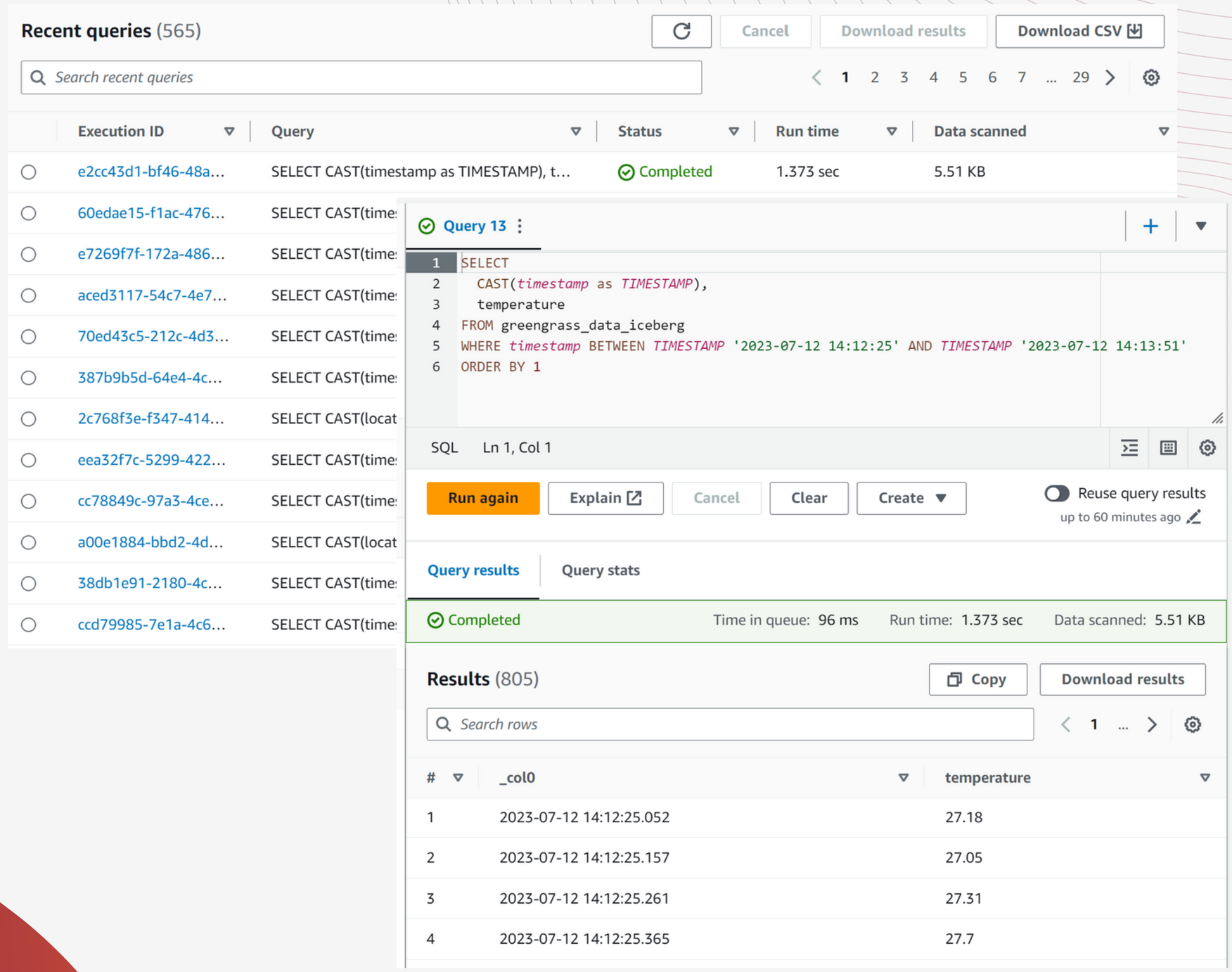
VISUALIZE THE DATA

- Managed Grafana
- Fully Managed Athena data source support
- Column autofill



VISUALIZE THE DATA

- Managed Grafana




aws
Search or jump to...
ctrl+k
Home › Dashboards › Robocat
2023-07-12 22:12:24 to 2023-07-12 22:13:51
Robocat Location
robocat-01
Robocat Speed
56
Robocat Speed
62
60
58
56
54
52
50
48
46
22:12:30
22:12:45
22:13:00
22:13:15
22:13:30
22:13:45
speed
Robocat Temp
32
30
28
26
24
22
20
18
22:12:30
22:12:40
22:12:50
22:13:00
22:13:10
22:13:20
22:13:30
22:13:40
22:13:50
temperature

WHERE CAN I GET ONE?

- All the bits are on GitHub
  - Data Pipeline
  - Edge components and sample code

https://github.com/t04glovern/aws-greengrass-bricks

WHAT DOES IT COST?

Breakdown the costs associated with my solution

Assumptions

- Data Size: 10TB (10,000GB) - Queried through Athena
- Batch Size: 100
- Incoming File Size (per record): 166.25 bytes
- Batched Incoming File Size (per 100 records): 16,625 bytes
- Iceberg File Size (configured): 512MB

Costs

- S3 PutObject (with batched size): **$3 (monthly)**
- Iceberg S3 GetObject (with 512MB file size): **$0.01 (monthly)**
- S3 data returned to Athena:
  - **$~7** returned, **$~20** scanned **(10 tb data)**
- Iceberg S3 Storage: **$230.00 (monthly)**
- Athena ($5 per tb): **$50 (10 tb data)**
  - Note that I haven't calculated for compression ratio here - $50 is the worst-case scenario.

Extras

- Amazon Managed Grafana: $5 per user per month.
- Lambda & SQS: $~10 (if that).

Not Pictured

- KMS: You probably need this, and it can be expensive
  - **Expect** KMS costs to scale with GetObject requests.

CONS

NOT "MANAGED"

- Iceberg Table is 90% managed - the last 10% is **essential** to get right.

SQL'Y

- Managing schema changes is going to require SQL wisdom
- Tools like Flyway or Alembic or anything you might use for managing automatic schema upgrades? - Non-existent.

NOT BATTLE HARDENED

- It's comparatively very new tech, and it shows - especially the Athena Iceberg variant.
- I with Athena supported COPY INTO iceberg from S3 - the way you can with Snowflake

PROS

PRICE CAN MAKE SENSE

- All pieces are usage-based billing.
- Kinesis and Glue DPU hourly costs can't hurt you here, you're safe.

RAW DATA

- Data lands in the purest form (that makes sense financially)
- Hive-style partitioning means you have an out if Iceberg doesn't work for you.

FLEXIBLE & RESILIENT

- Failures can be reconciled by simply reprocessing events.
- Changes can be made with confidence

CONTACT ME

nathan@glovers.id.au

devopstar.com

@nathangloverAUS

THANK YOU

FOR LISTENING