
*BIG DATA CASE STUDY NIGHT #3*

# STREAM AND ANALYZE TIME SERIES IN REAL-TIME WITH AWS

**John Mousa**

Sr. Solutions Architect
Amazon Web Services

# John Mousa

Engineer at heart, enabling enterprise customers in DACH and world wide bridge the worlds of microservices and analytics as a Sr. Solutions Architect at Amazon Web Services.

/in/johnmousa

/johnmousa

@JohnMousa_

# Agenda for Today

Our use case for Power and Utilities

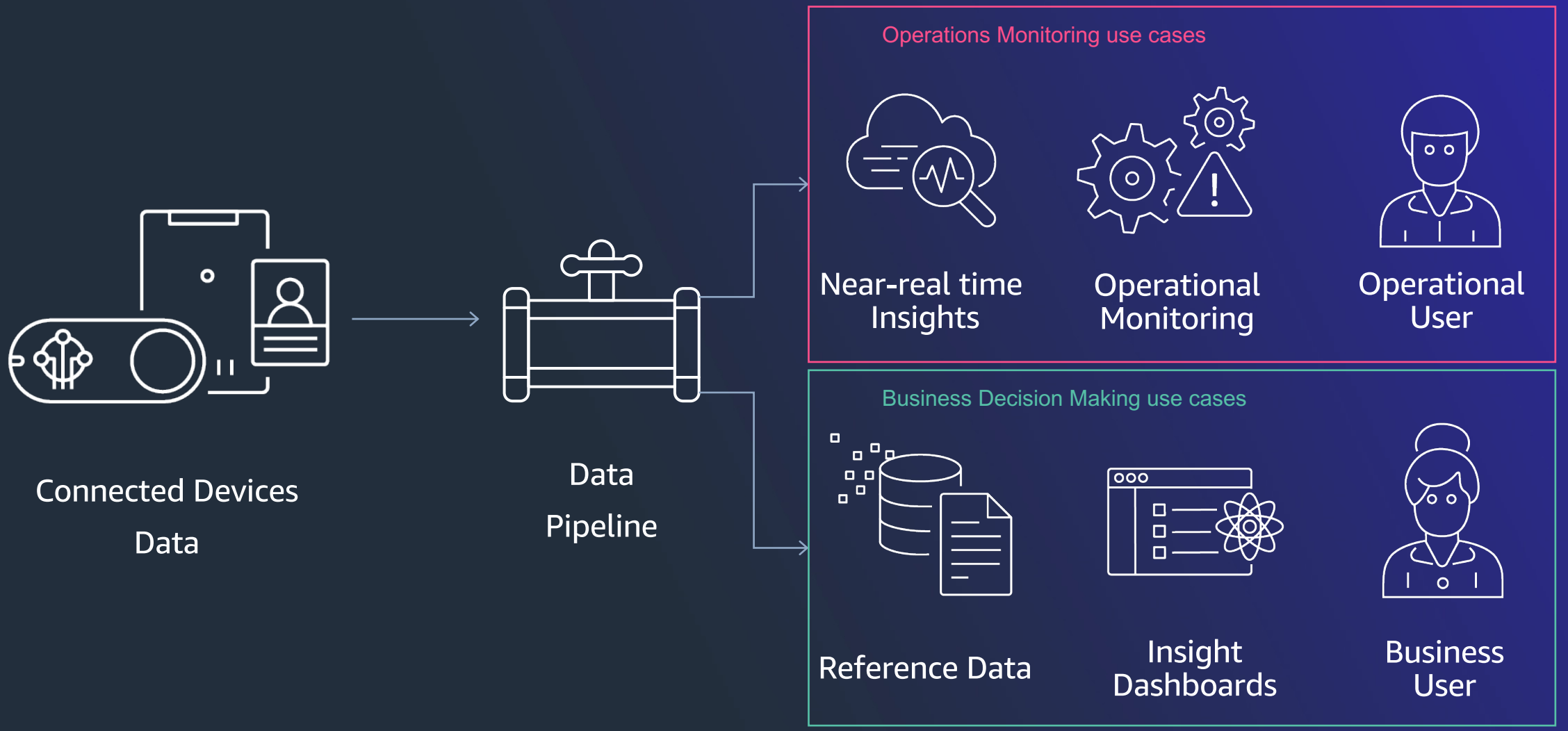
Overview on Architecture

Solutions

Operational View

Strategic View

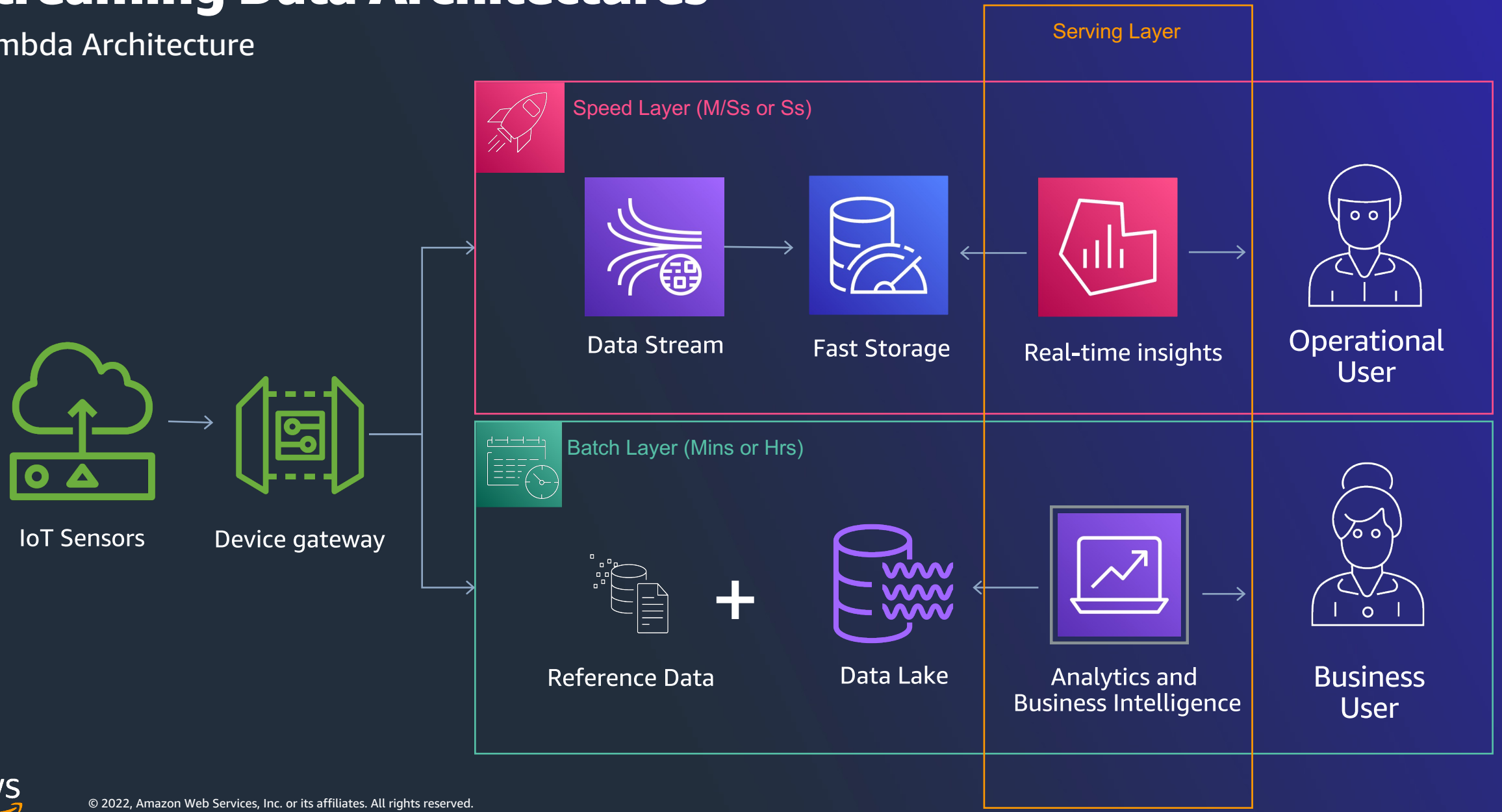
# Collecting Telemetry Data from Connected Devices



# Architectures

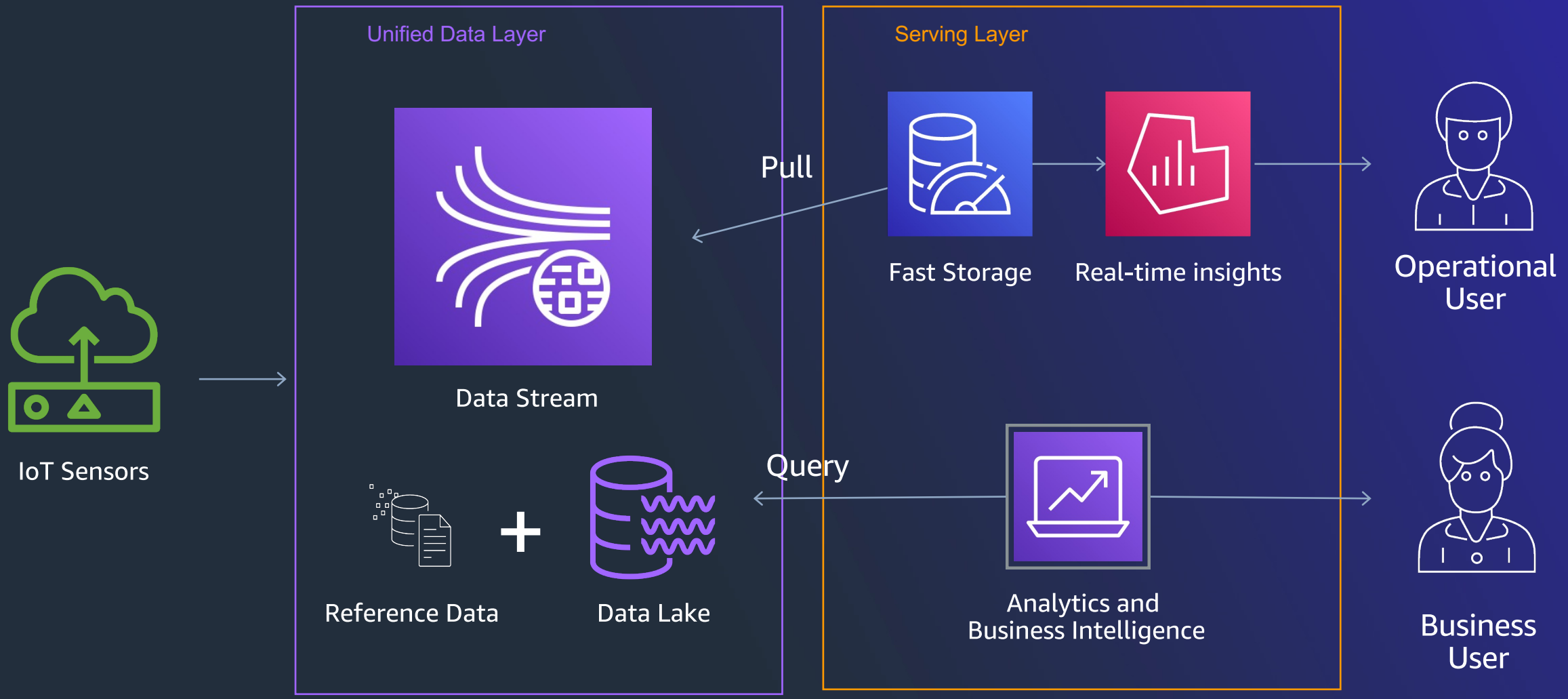
# Streaming Data Architectures

## Lambda Architecture



# Streaming Data Architectures

## Kappa Architecture



# Streaming Data Architectures

Lambda VS Kappa

λ

- Batch layer is built for big data
- Result in overhead due to code and resource duplication

K

- No redundancies
- Batch workloads needs to be in a sink

# Solutions

# But Why Did We Consider Such Solutions

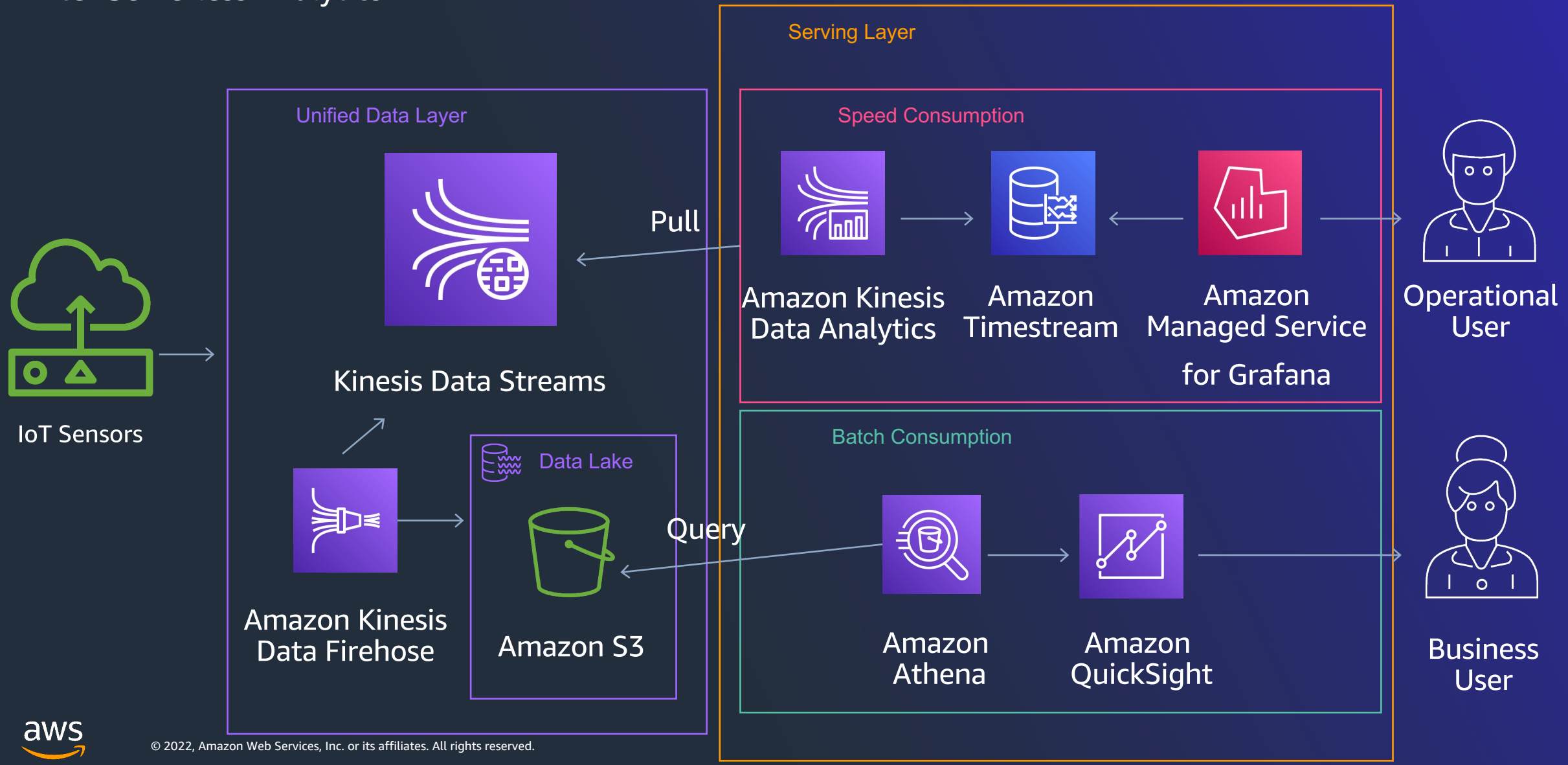
Streaming is Expensive

Streaming is Lossy

Hard to Operate,
Scale and Maintain

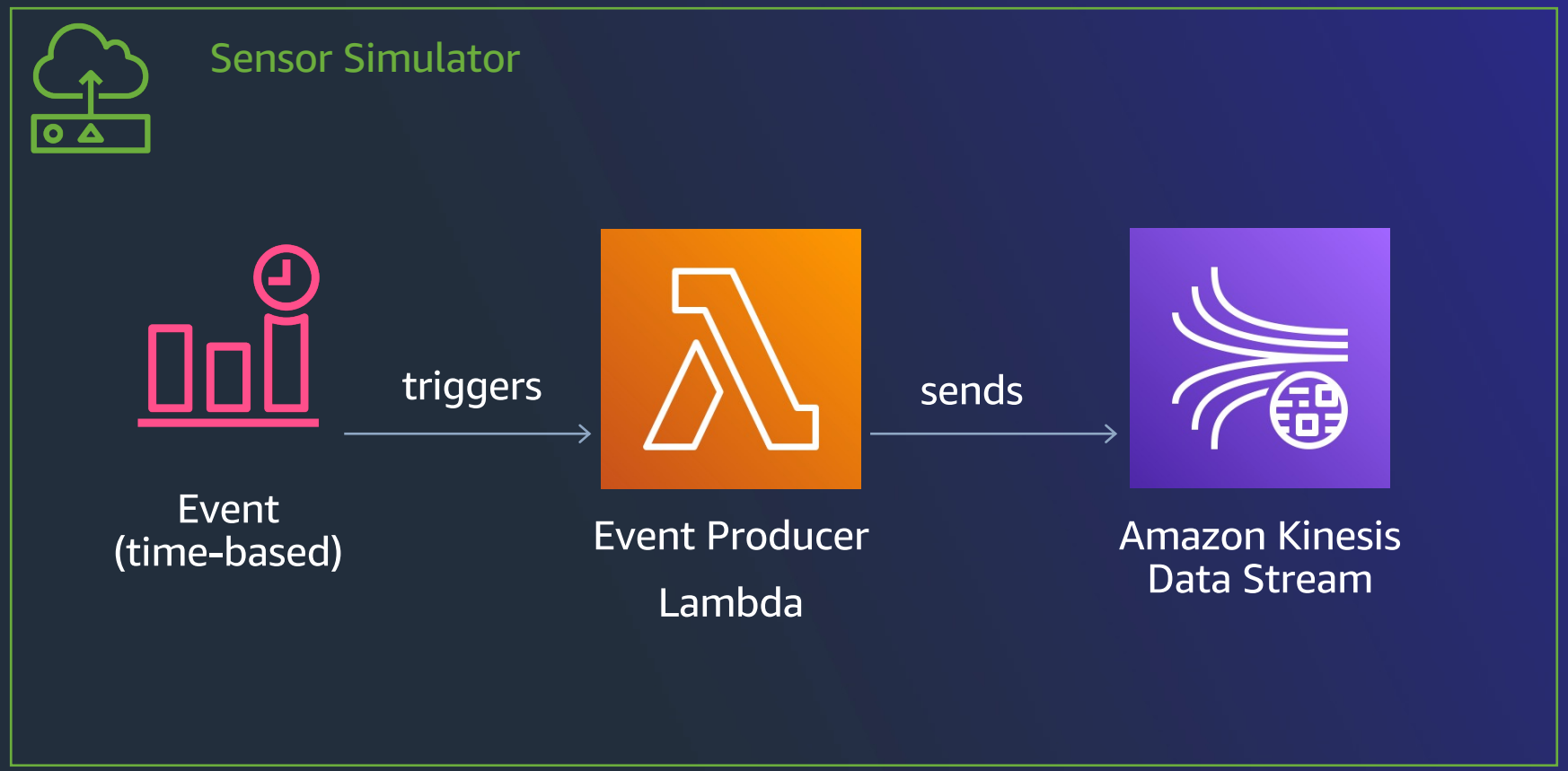
# Current Data Architecture

Enter Serverless Analytics



# Ingestion

# Data Ingestion from Devices



# Data Ingestion from Devices

## CDK Infrastructure

```
sample_device_producer = aws_lambda_python.PythonFunction(self, 'SampleDeviceProducer',
                                                           entry='stacks/sample_kinesis_stream_producer/producer_lambda',
                                                           index='app.py',
                                                           runtime=aws_lambda.Runtime.PYTHON_3_8,
                                                           timeout=core.Duration.seconds(30))

stream.grant_write(sample_device_producer)

lambda_input = {"Stream": stream.stream_name}
Rule(self, 'ProducerTriggerEventRule',
     enabled=True,
     schedule=Schedule.rate(Duration.minutes(1)),
     targets=[aws_events_targets.LambdaFunction(handler=sample_device_producer,
                                                event=RuleTargetInput.from_object(lambda_input))])
```

# Data Ingestion from Devices

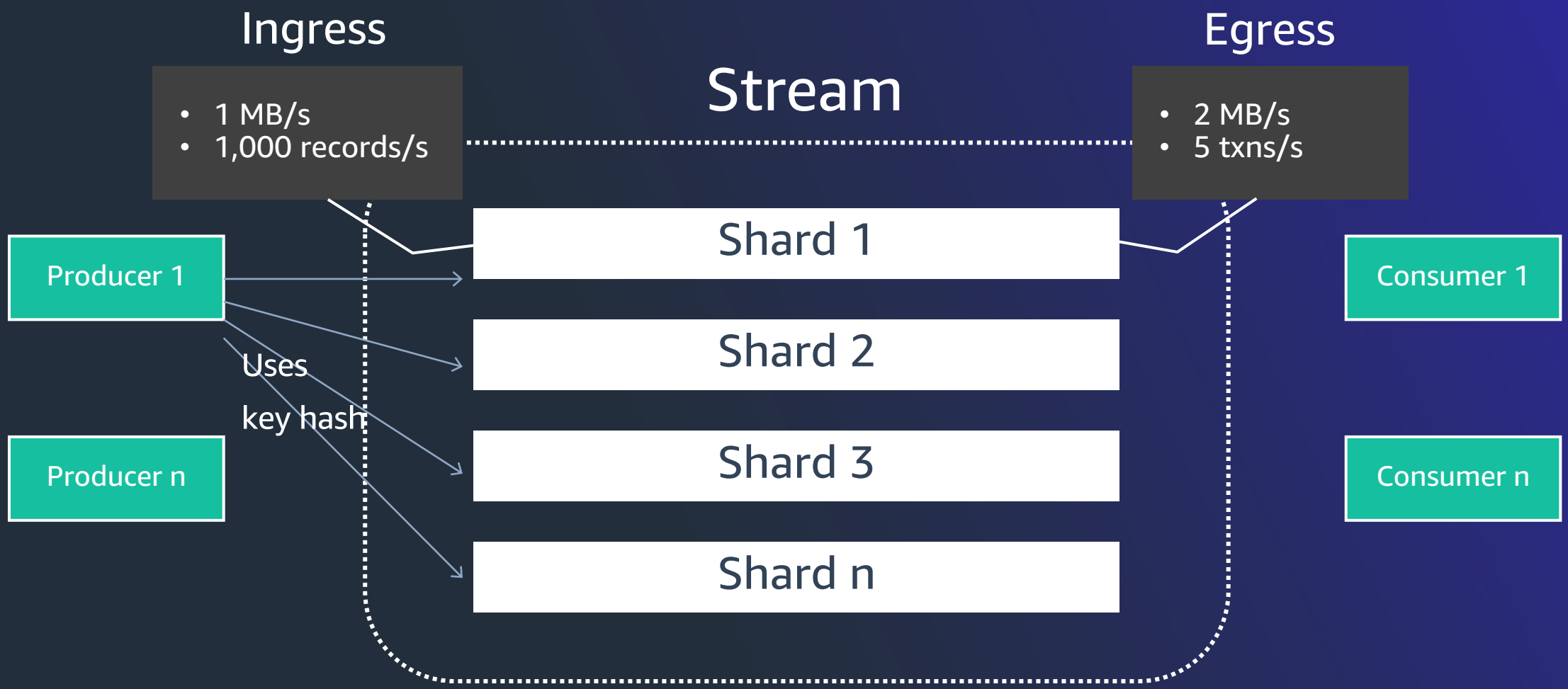
## Lambda Writing to Kinesis Data Streams

```python
def write_records(stream_name, records):
    kinesis_records = []
    for record in records:
        kinesis_records.append({
            'Data': json.dumps(record),
            'PartitionKey': record["DeviceID"]
        })

    result = None
    try:
        result = kinesis.put_records(
            StreamName=stream_name,
            Records=kinesis_records, )

        status = result['ResponseMetadata']['HTTPStatusCode']
        print("Processed %d records. WriteRecords Status: %s" %
              (len(records), status))
    except Exception as err:
        print("Error:", err)
        if result is not None:
            print("Result:{}".format(result))
```

```python
def handler(event, context):
    sent = 0
    for x in range(iterations):
        records = []
        for device_id in device_ids:
            records.append(prepare_record(measures, device_id,
                                          (iterations - x) / iterations * 60))

        print("records {}".format(len(records)))
        write_records(event["Stream"], records)
        sent = sent + len(records)
    return {"records": sent}
```

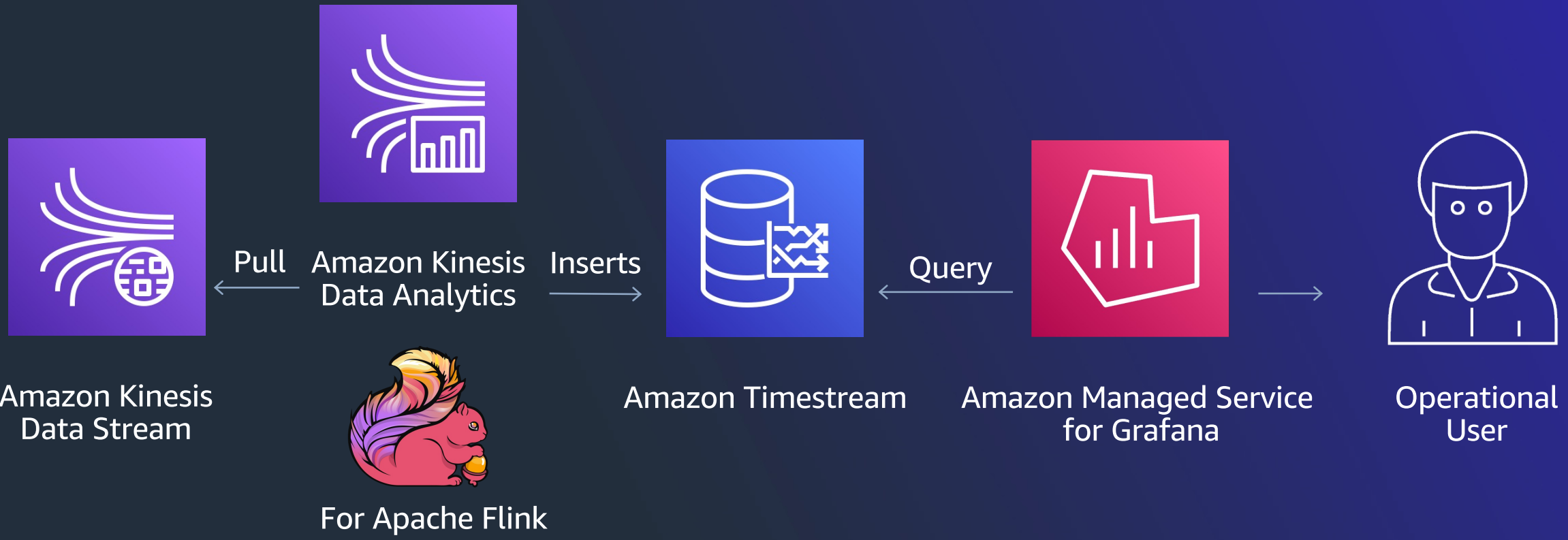
# Data Streaming

# Kinesis Data Streams: Granular Provisioning and Scaling



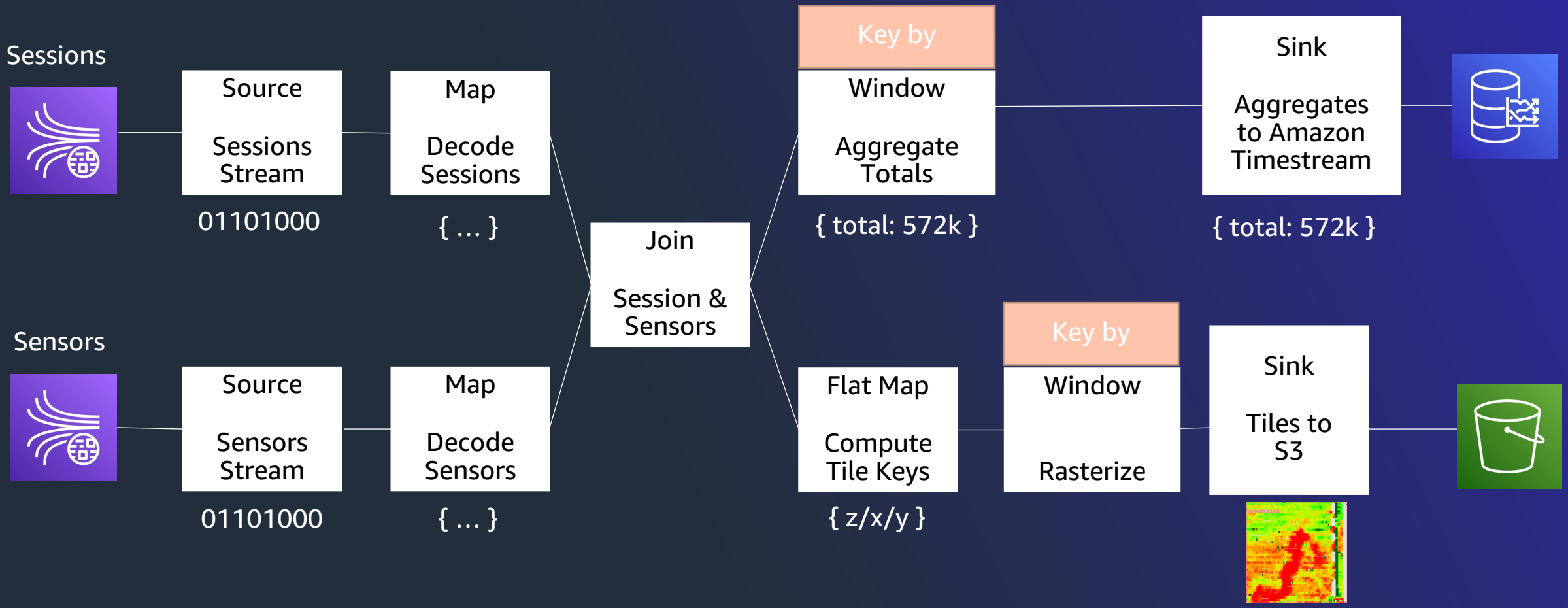
# Operational View

# Near-Real Time Operational Insights Architecture



# Streaming Analytics

Apache Flink - Example



# Streaming Analytics

Apache Flink – Declarative Streams

```
createKinesisSource(env, parameter)
    .map(JsonToTimestreamPayloadFn()).name("MaptoTimestreamPayload")
    .process(OffsetFutureTimestreamPoints()).name("UpdateFutureOffsetedTimestreamPoints")
    .addSink(TimestreamSink(region, databaseName, tableName, batchSize))
    .name("TimestreamSink<$databaseName, $tableName>")
```

# Amazon Timestream Architecture

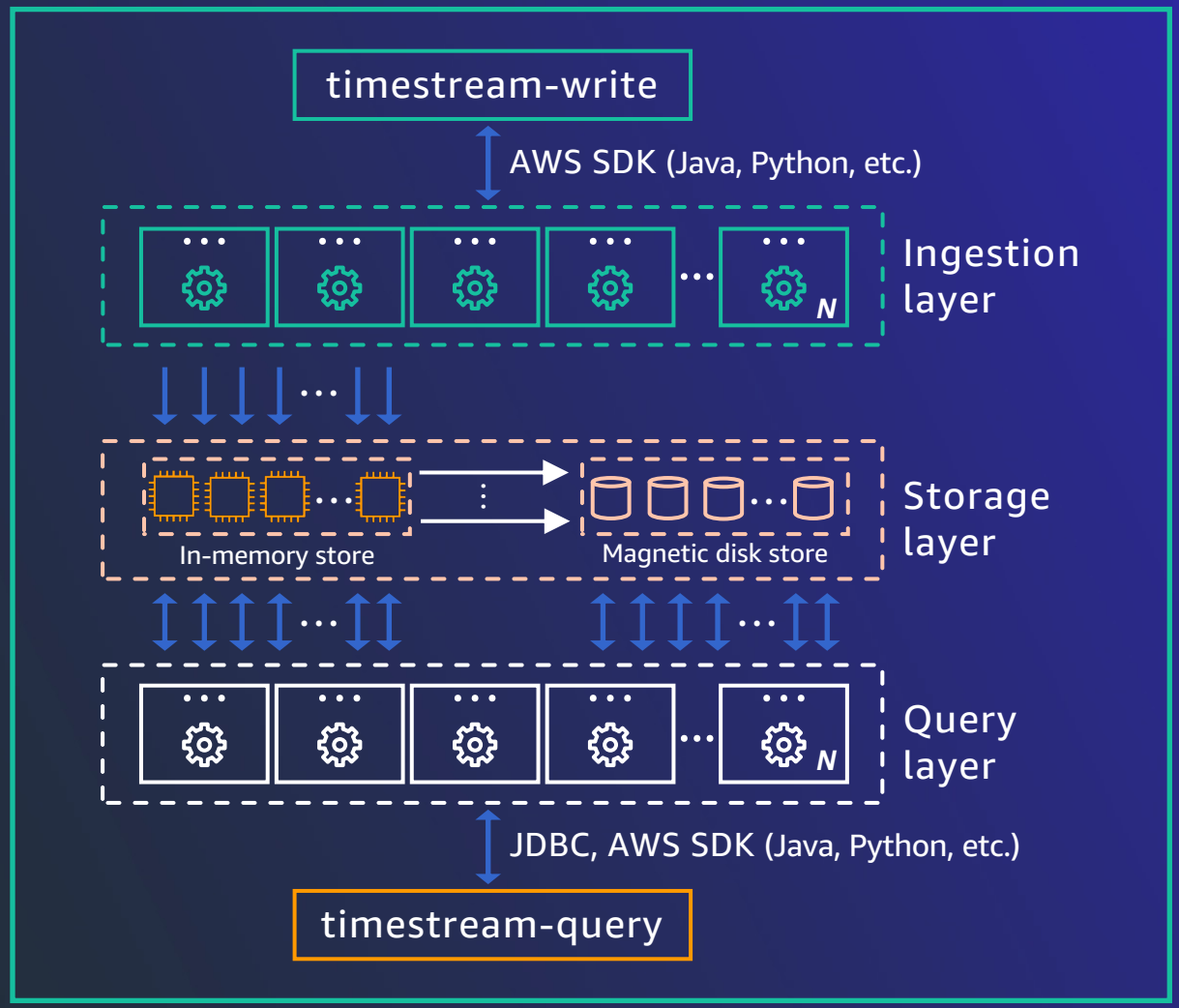
Purpose-built for time series workloads

## Decoupled architecture

- Highly available
- ***Independently*** scalable ingestion, storage, and query processing

## High throughput auto scaling ingestion

- ***Durable***: data is ***replicated*** across multiple Availability Zones
- Automatic data deduplication handling
- No need to provision or configure resources



# Amazon Timestream Architecture

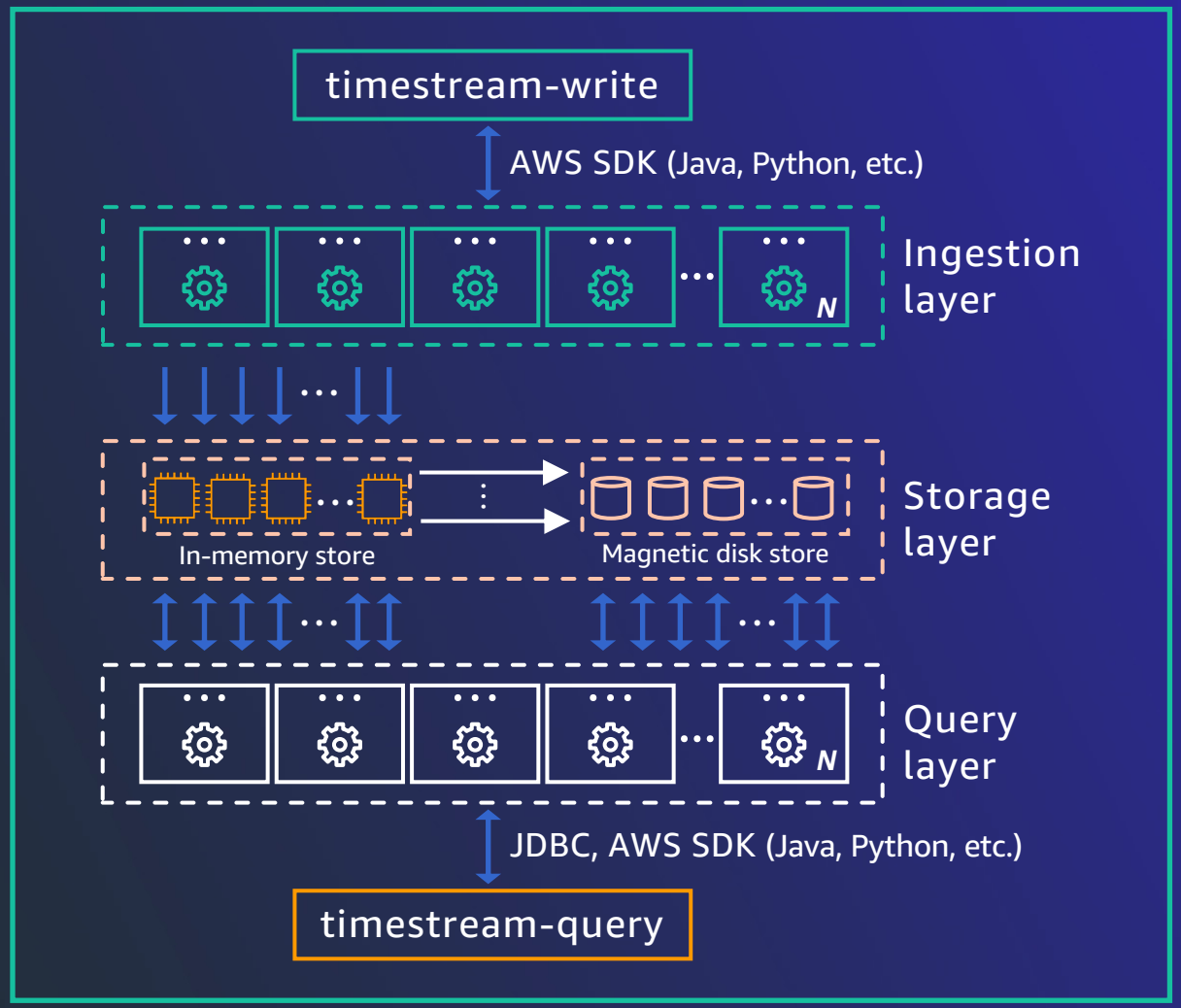
Serverless, cloud-native AWS built technology

## Multiple tiers of storage

- Scalable to petabytes and beyond
- ***Memory store***: fast point-in-time queries, high-throughput durable ingestion
- ***Magnetic store***: high-performance analytical queries and low-cost long-term storage

## Scalable SQL query engine

- ***Adaptive query engine*** is capable of querying data across multiple data tiers
- ***No indexes*** to configure and ***no provisioning*** required

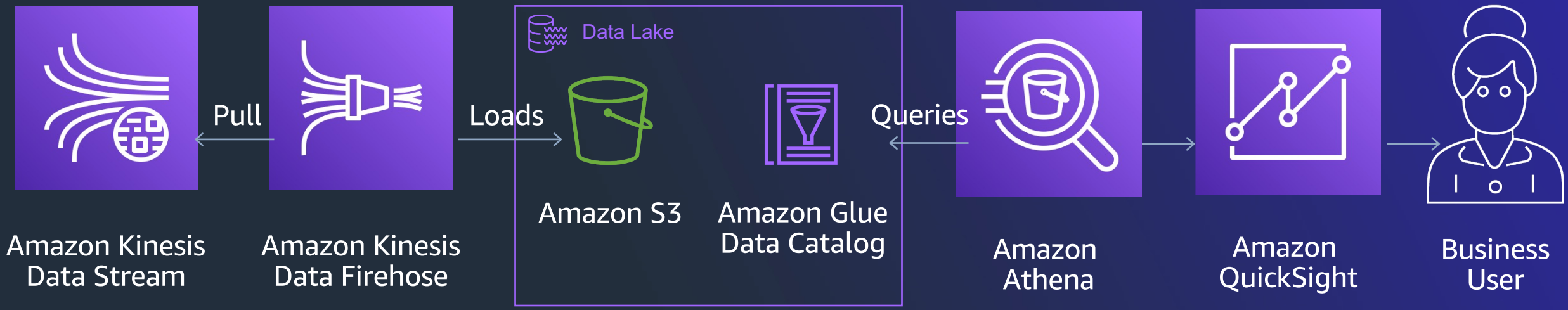


# Operational Views

## Amazon Managed Service for Grafana

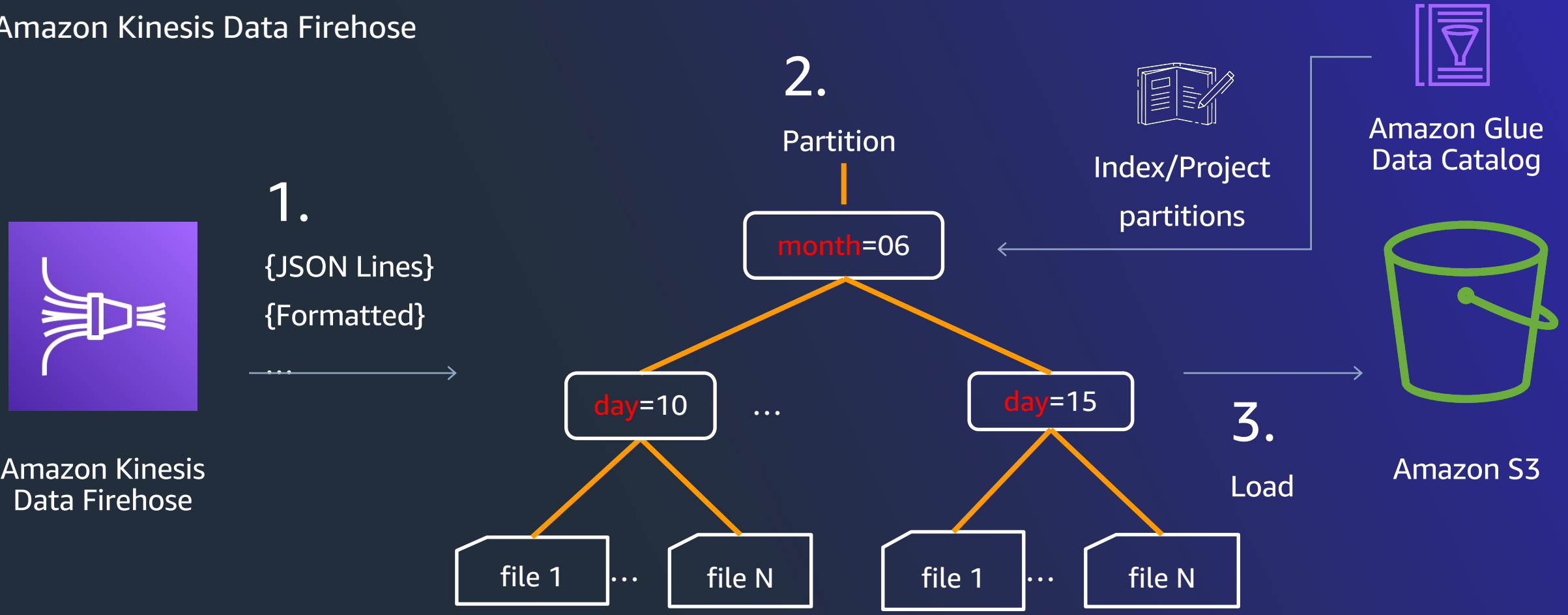
# Strategic Big Data Insights

# Strategic Big Data Insights Architecture



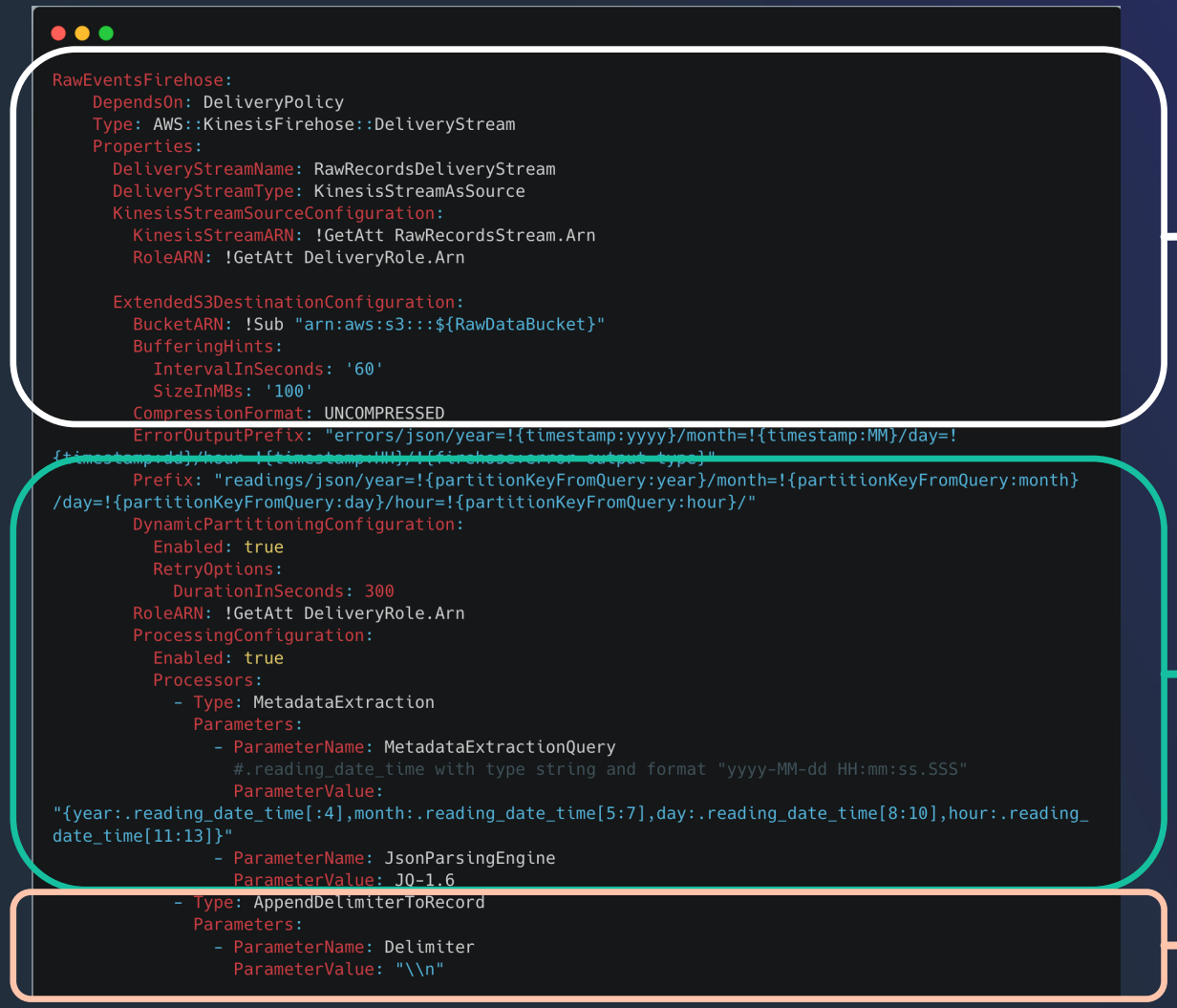
# Data Encoding and Partitioning

Amazon Kinesis Data Firehose



# Data Encoding and Partitioning

## Amazon Kinesis Data Firehose



```
RawEventsFirehose:
    DependsOn: DeliveryPolicy
    Type: AWS::KinesisFirehose::DeliveryStream
    Properties:
      DeliveryStreamName: RawRecordsDeliveryStream
      DeliveryStreamType: KinesisStreamAsSource
      KinesisStreamSourceConfiguration:
        KinesisStreamARN: !GetAtt RawRecordsStream.Arn
        RoleARN: !GetAtt DeliveryRole.Arn

      ExtendedS3DestinationConfiguration:
        BucketARN: !Sub "arn:aws:s3:::${RawDataBucket}"
        BufferingHints:
          IntervalInSeconds: '60'
          SizeInMBs: '100'
        CompressionFormat: UNCOMPRESSED
        ErrorOutputPrefix: "errors/json/year=!{timestamp:yyyy}/month=!{timestamp:MM}/day=!
{timestamp:dd}/hour=!{timestamp:HH}/!{firehose:error-output-type}"
        Prefix: "readings/json/year=!{partitionKeyFromQuery:year}/month=!{partitionKeyFromQuery:month}
/day=!{partitionKeyFromQuery:day}/hour=!{partitionKeyFromQuery:hour}/"
        DynamicPartitioningConfiguration:
          Enabled: true
          RetryOptions:
            DurationInSeconds: 300
        RoleARN: !GetAtt DeliveryRole.Arn
        ProcessingConfiguration:
          Enabled: true
          Processors:
            - Type: MetadataExtraction
              Parameters:
                - ParameterName: MetadataExtractionQuery
                  #.reading_date_time with type string and format "yyyy-MM-dd HH:mm:ss.SSS"
                  ParameterValue:
"{year:.reading_date_time[:4],month:.reading_date_time[5:7],day:.reading_date_time[8:10],hour:.reading_
date_time[11:13]}"
                - ParameterName: JsonParsingEngine
                  ParameterValue: JQ-1.6
            - Type: AppendDelimiterToRecord
              Parameters:
                - ParameterName: Delimiter
                  ParameterValue: "\\n"
```

Move data from Stream to Raw S3 Bucket

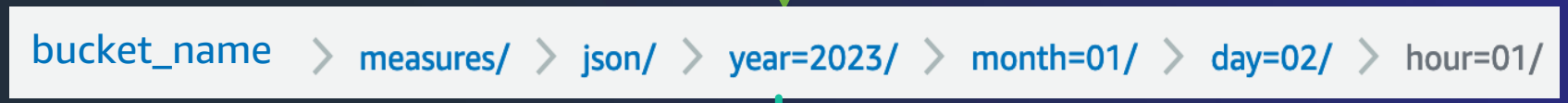
Late arriving data compensation

JSON lines

# Discover Dynamic Partitions

Glue Data Catalog Partition Projection

```
        Prefix: "readings/json/year=!{partitionKeyFromQuery:year}/month=!{partitionKeyFromQuery:month}
/day=!{partitionKeyFromQuery:day}/hour=!{partitionKeyFromQuery:hour}/"
```

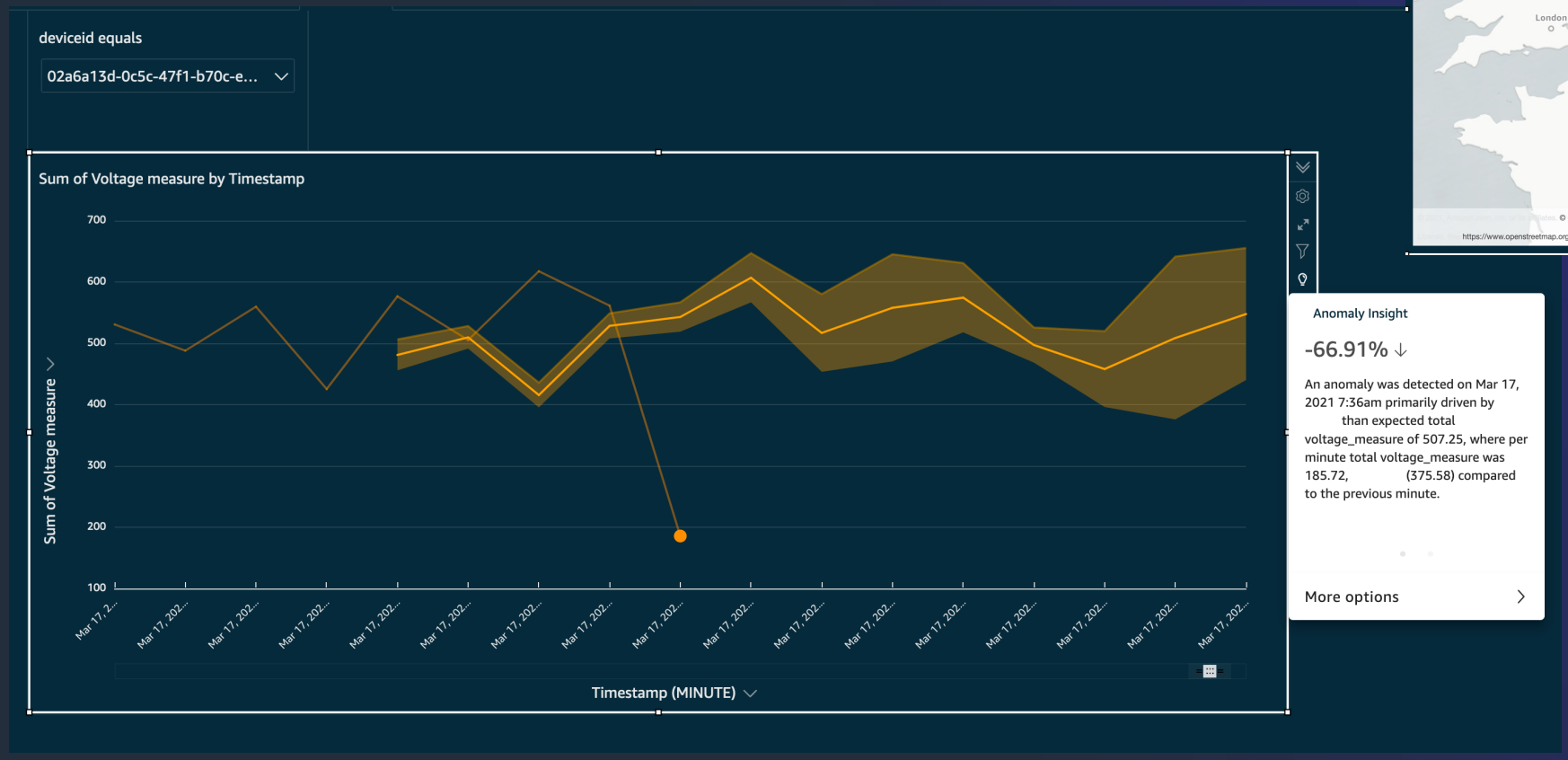


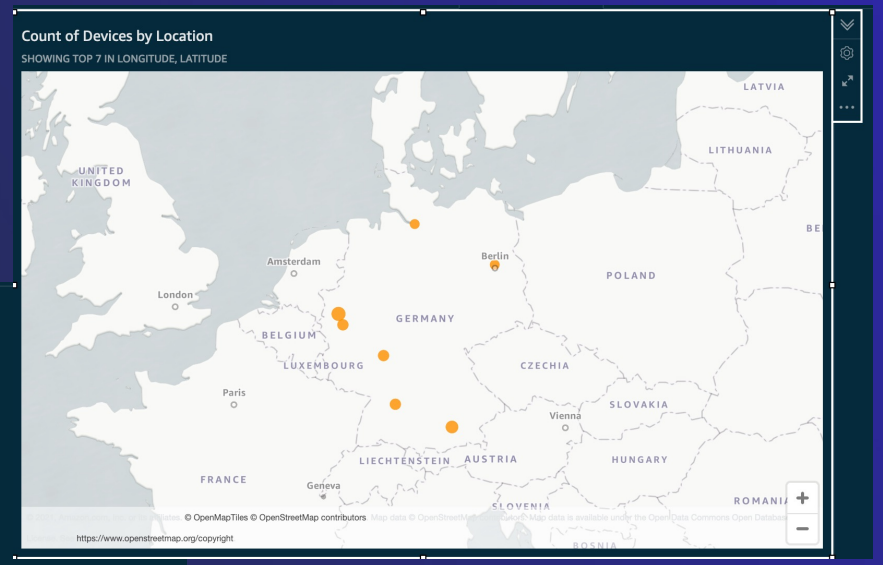

```
CREATE EXTERNAL TABLE raw_device_data (
 ...
)
...
PARTITIONED BY (
 year  INT,
 month INT,
 day   INT,
 hour  INT,
)
LOCATION "s3://DOC-EXAMPLE-BUCKET/prefix/"
TBLPROPERTIES (
 "projection.enabled" = "true",
 "projection.year.type" = "integer",
 "projection.year.range" = "2011,2099",
 "projection.year.digits" = "4",
 "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://DOC-EXAMPLE-BUCKET/prefix/year=${year}/month=${month}/day=${day}
/hour=${hour}/"
)
```

```
"projection.enabled" = "true",
"projection.year.type" = "integer",
"projection.year.range" = "2011,2099",
"projection.year.digits" = "4",
```

# Insights Views

## Amazon QuickSight



# Cost Factors

## Dynamic (per load)

- Athena when queried / GB scanned
- Kinesis
  - Data ingested, per GB (Includes 24-hour retention) $0.08
  - Data retrievals, per GB $0.04
- Kinesis Firehose
- Amazon Timestream
  - 1 million write of 1KB size $0.50

## Static (even when no events come)

- Amazon S3 Storage
- Amazon Timestream
  - Price per GB stored per hour $0.036
- Amazon Kinesis
  - Per stream, per hour $0.04
- Kinesis Data Analytics
  - Kinesis Processing Unit, Per Hour $0.11 per hour
  - Running Application Storage, Per GB-month $0.10 per GB-month

* Cost as of 28th of September 2022, in US-EAST-1 region, you pay for what you actually consume, checkout https://calculator.aws

# Purpose-Built AWS Data Analytics Services for Your Modern Data Architecture

- Work backwards from context and use cases, don't build technical platforms
- Use the right tool for your use case
- AWS has multiple serverless analytics services for different needs
- You're one Experiment away

# More Resources

- Near real-time processing with Amazon Kinesis, Amazon Timestream, and Grafana Blog Post
- Companion samples github repository

- Patterns for IoT time series ingestion Blog Post
- Smart meter analytics QuickStart

# Please complete the session survey

Direct link to slides right after 5-click survey feedback

https://eventbox.dev/survey/SQ1X96F

# Thank you!

**John Mousa**

Amazon Web Services
Sr. Solutions Architect | AWS DACH Enterprise

/in/johnmousa

/johnmousa

@JohnMousa_