Streaming analytics for time-series data

And hopefully we don’t choke

John Mousa

Sr. Solutions Architect, Digital Native Businesses
Amazon Web Services

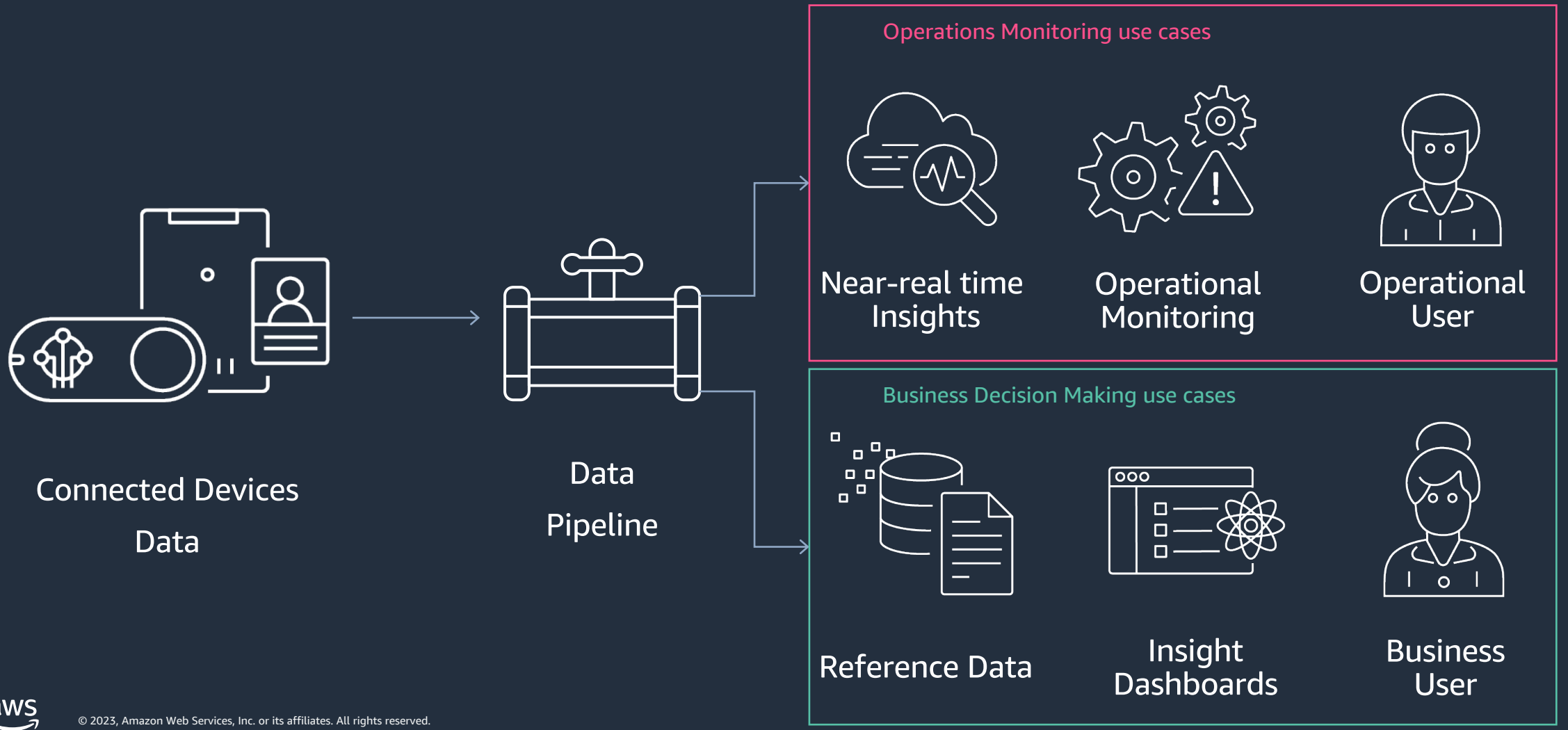
John Mousa

Engineer at heart, enabling customers in DACH and world wide to bridge the worlds of microservices and analytics

/in/johnmousa

/johnmousa

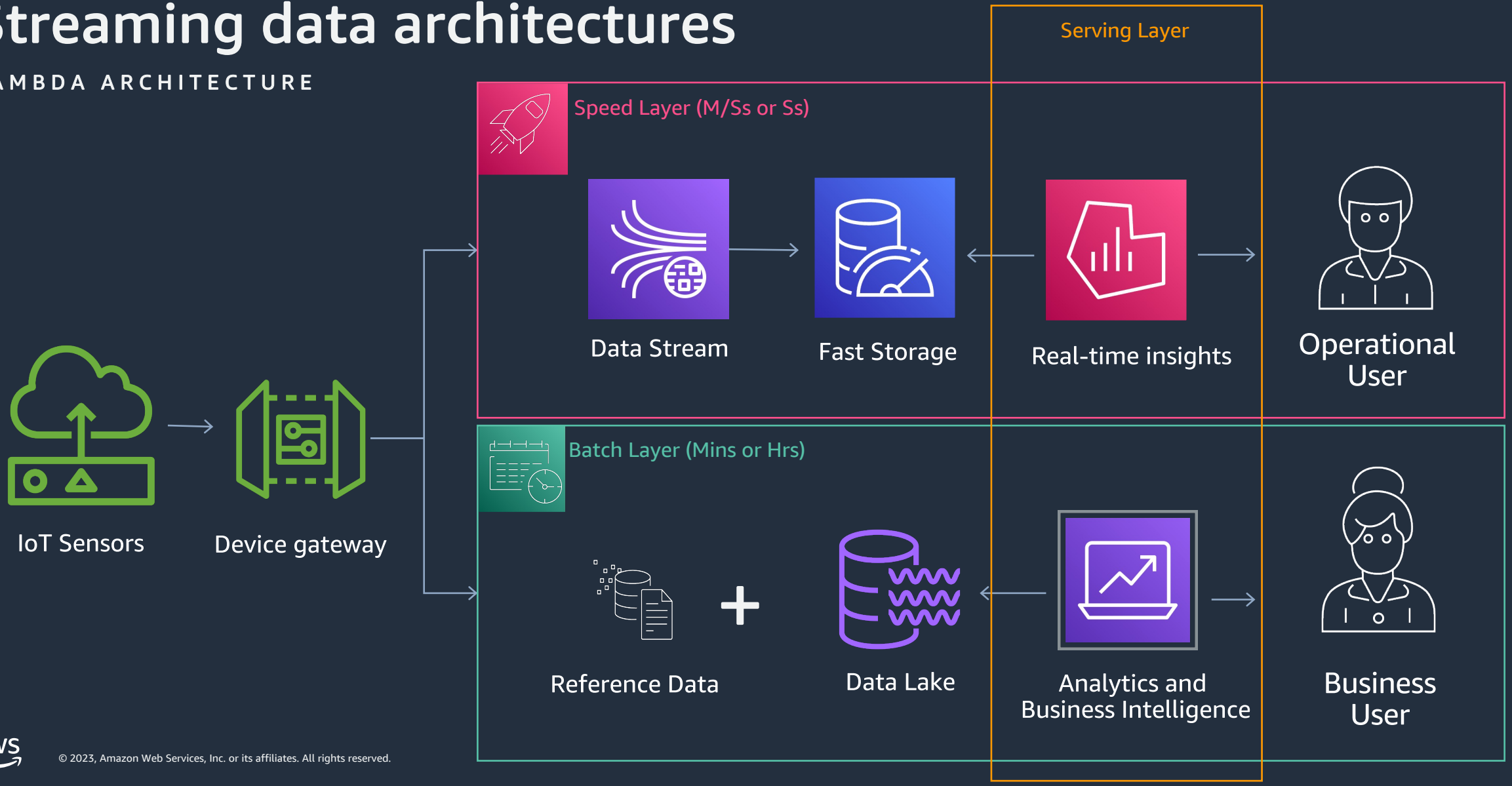
Collecting telemetry data from smart meters



Architecture

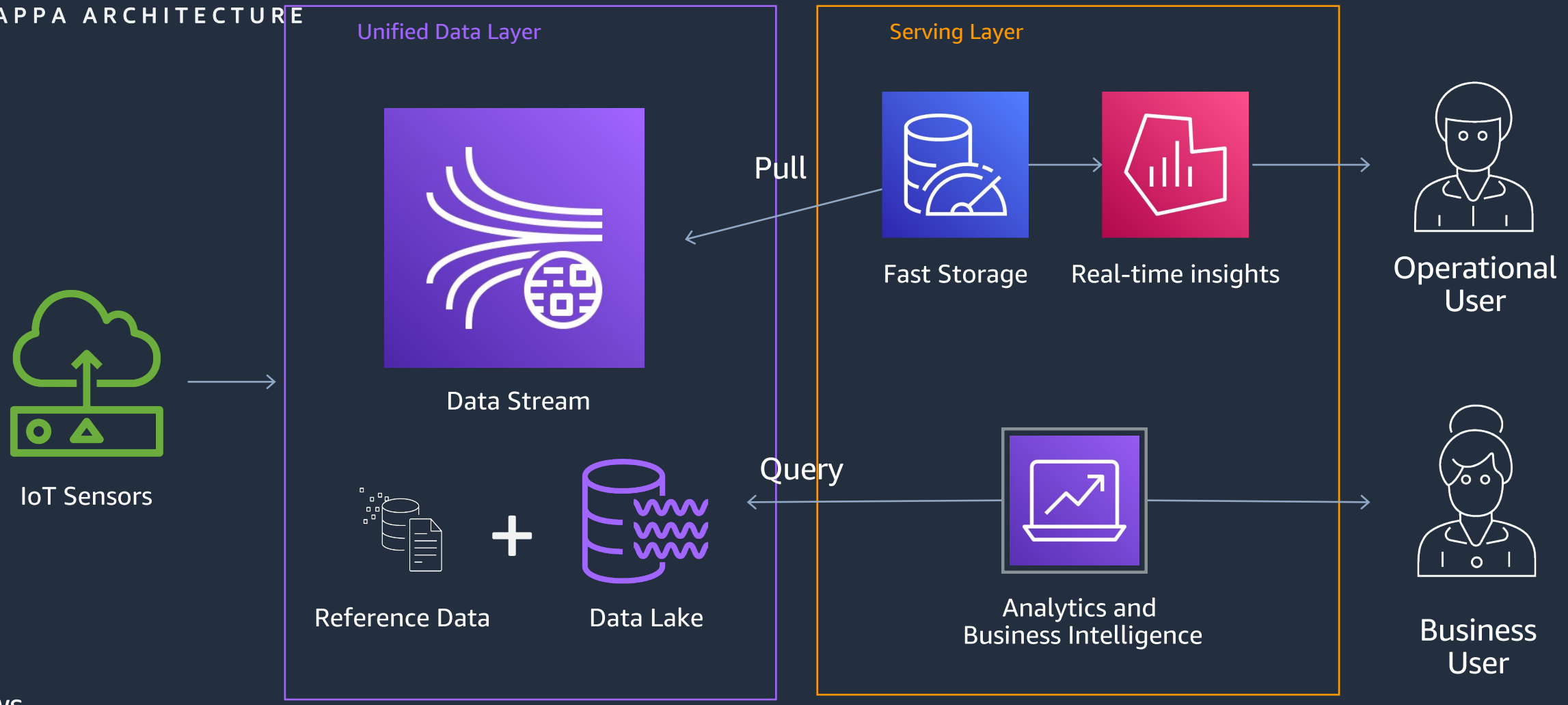
Streaming data architectures

LAMBDA ARCHITECTURE



Streaming data architectures

KAPPA ARCHITECTURE



Streaming data architectures

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

Streaming is Expensive

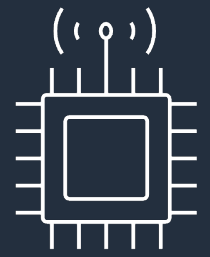
Streaming is Lossy

Hard to Operate,
Scale and Maintain

Both architectures are not designed to solve challenges

STREAMING ANALYTICS FOR TIME-SERIES DATA

This is really necessary evil



Smart meters

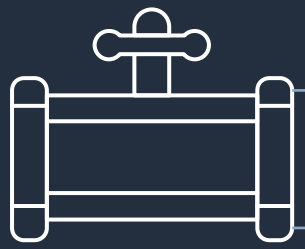
AWS IoT Services

Custom IoT Gateways

Regulator or customer gateway

Legacy batch meter Gateways

This is only one part of your architecture



Data Pipeline

Value is really here, we need to focus here

Operations Monitoring use cases

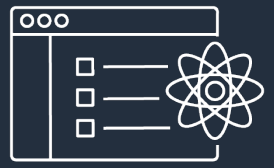
Near-real time Insights

Operational Monitoring

Operational User

Business Decision Making use cases

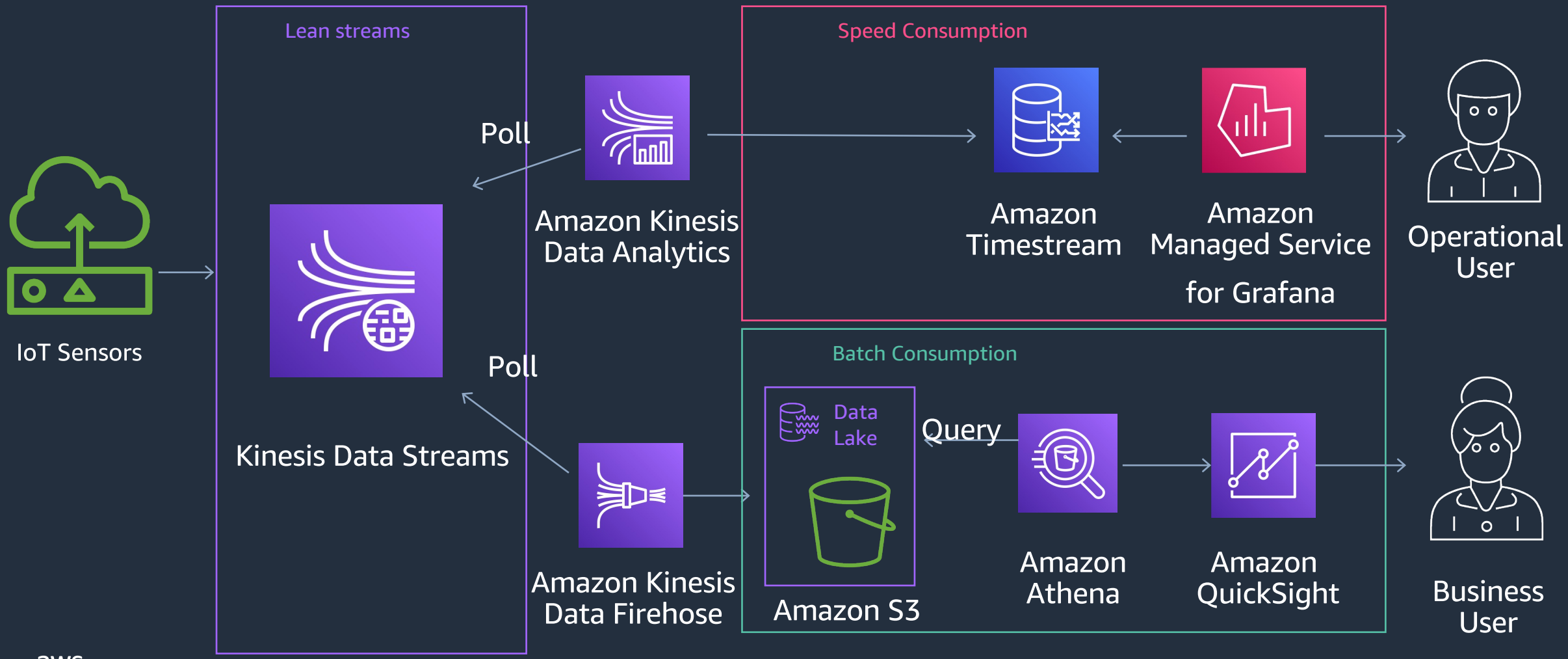
Reference Data



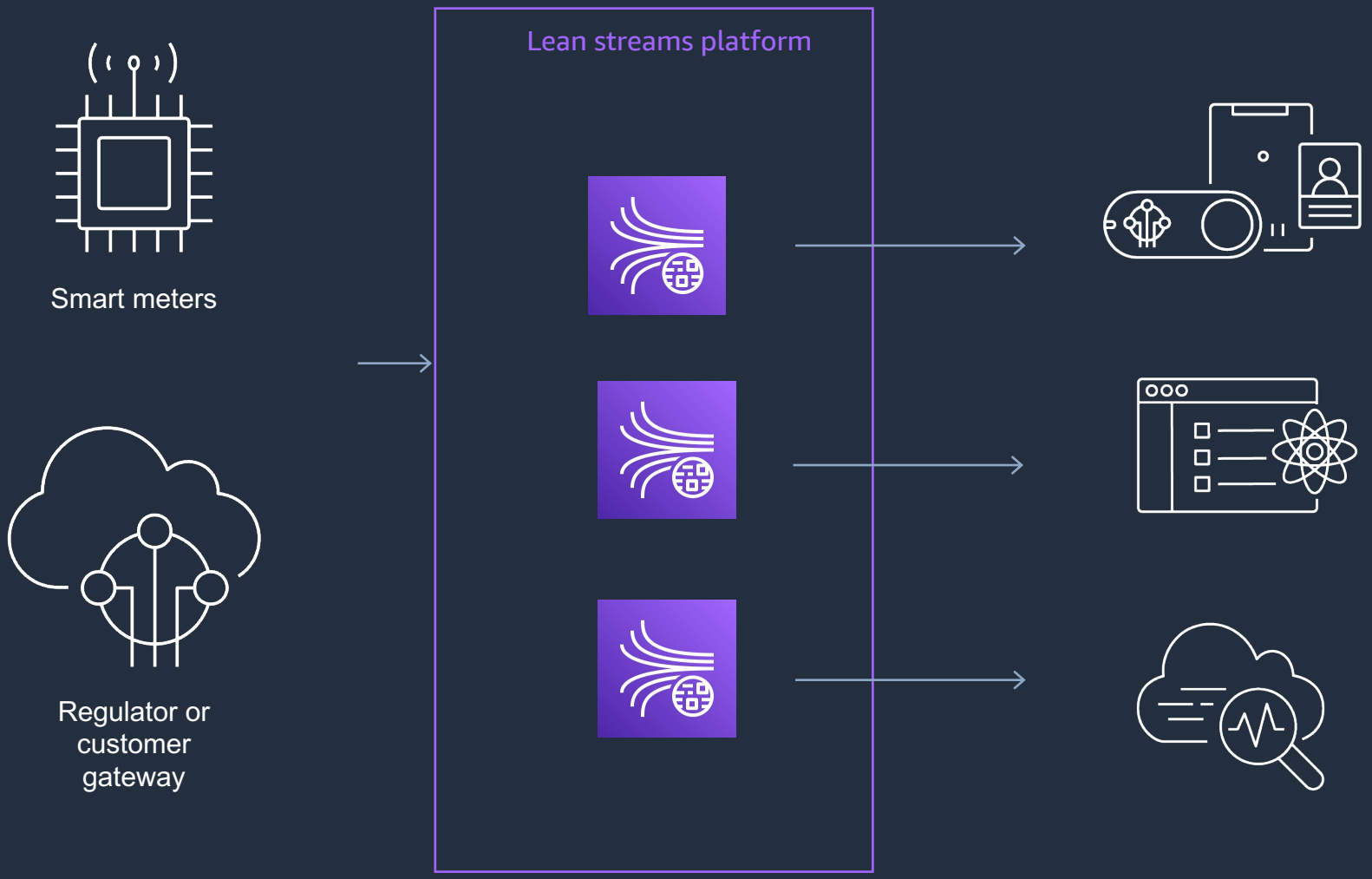
Insight Dashboards

Business User

Working backwards

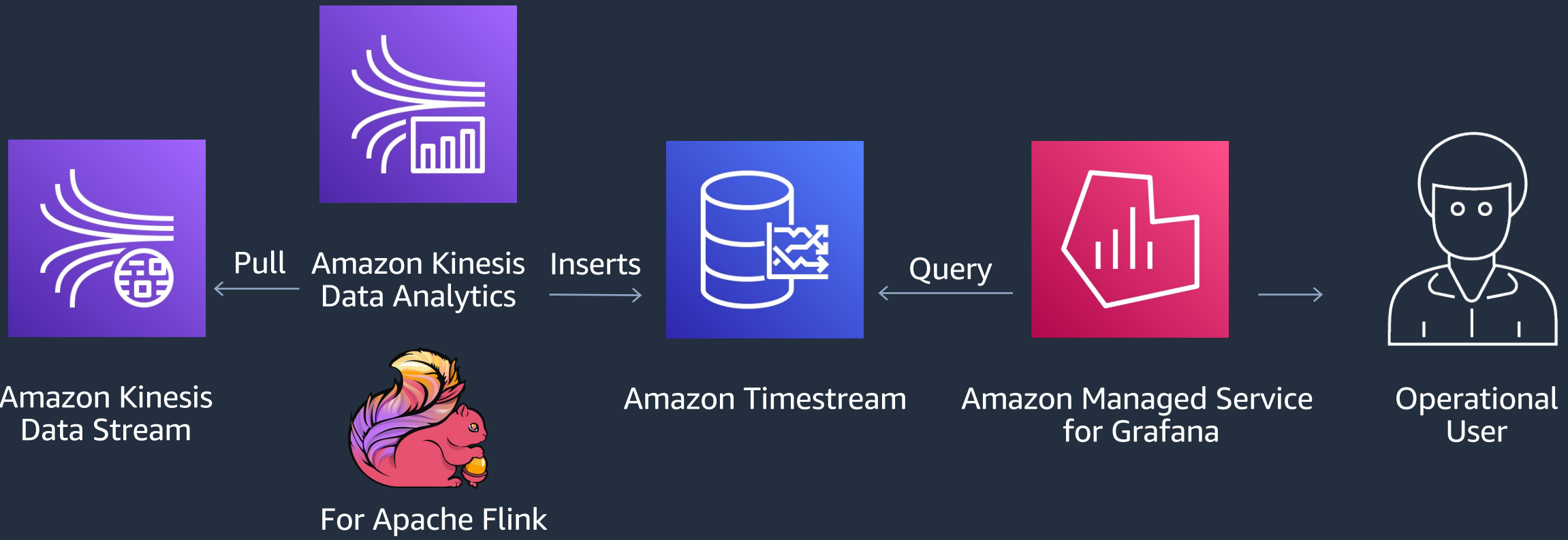


Working backwards



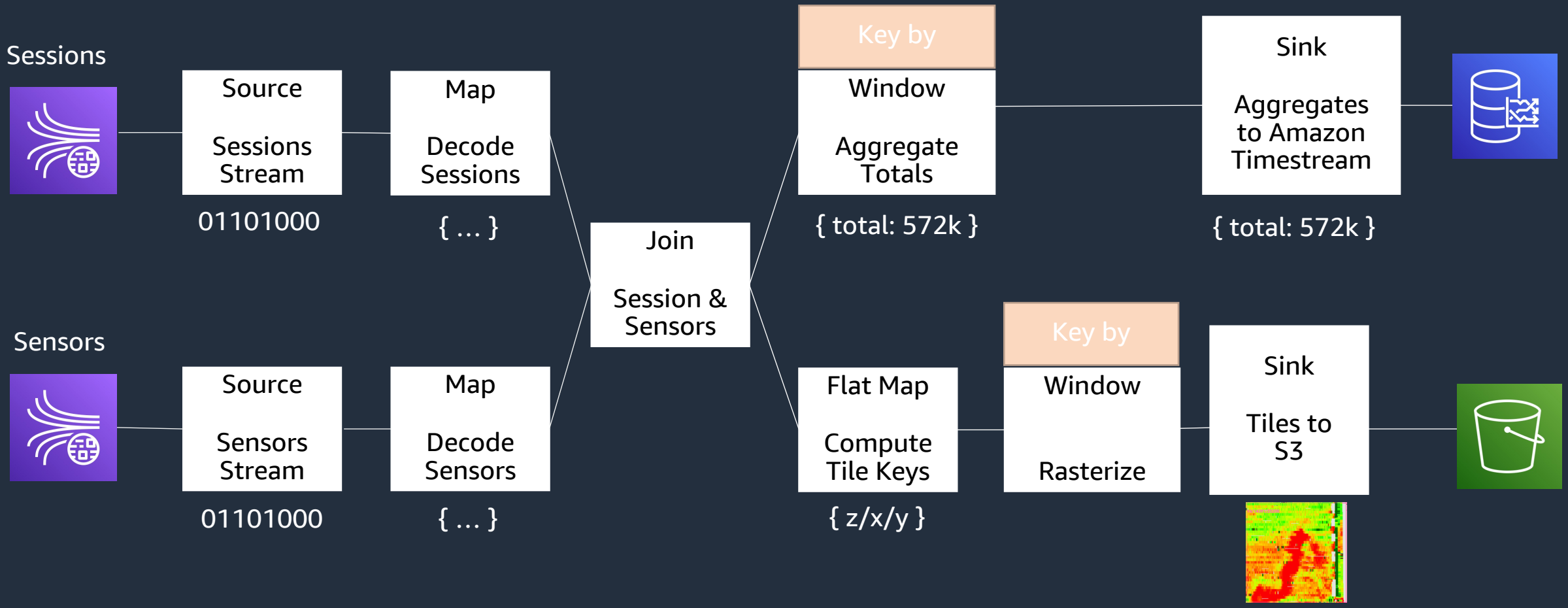
Speed consumption

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

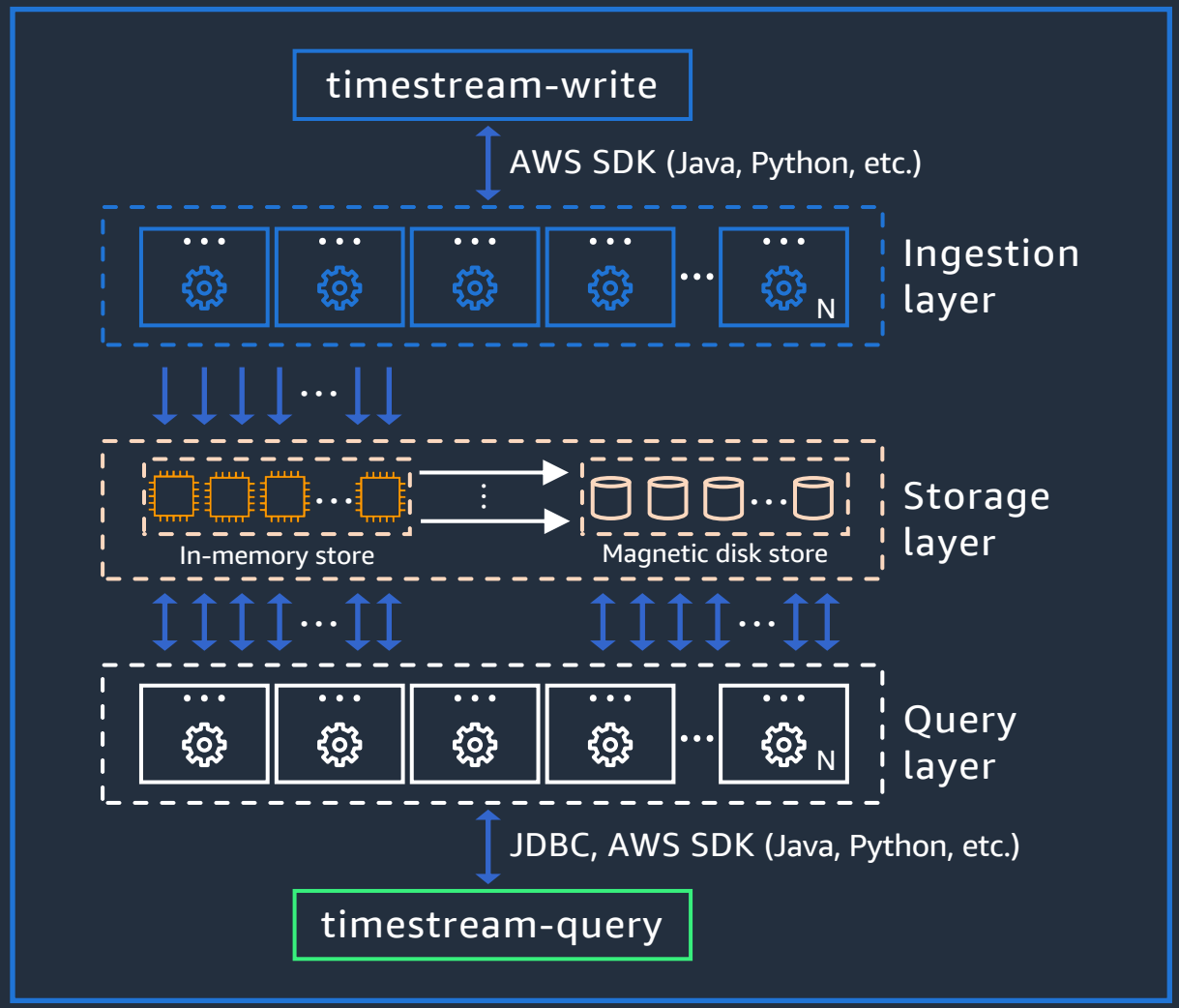
PURPOSE-BUILT FOR TIME SERIES WORKLOADS

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

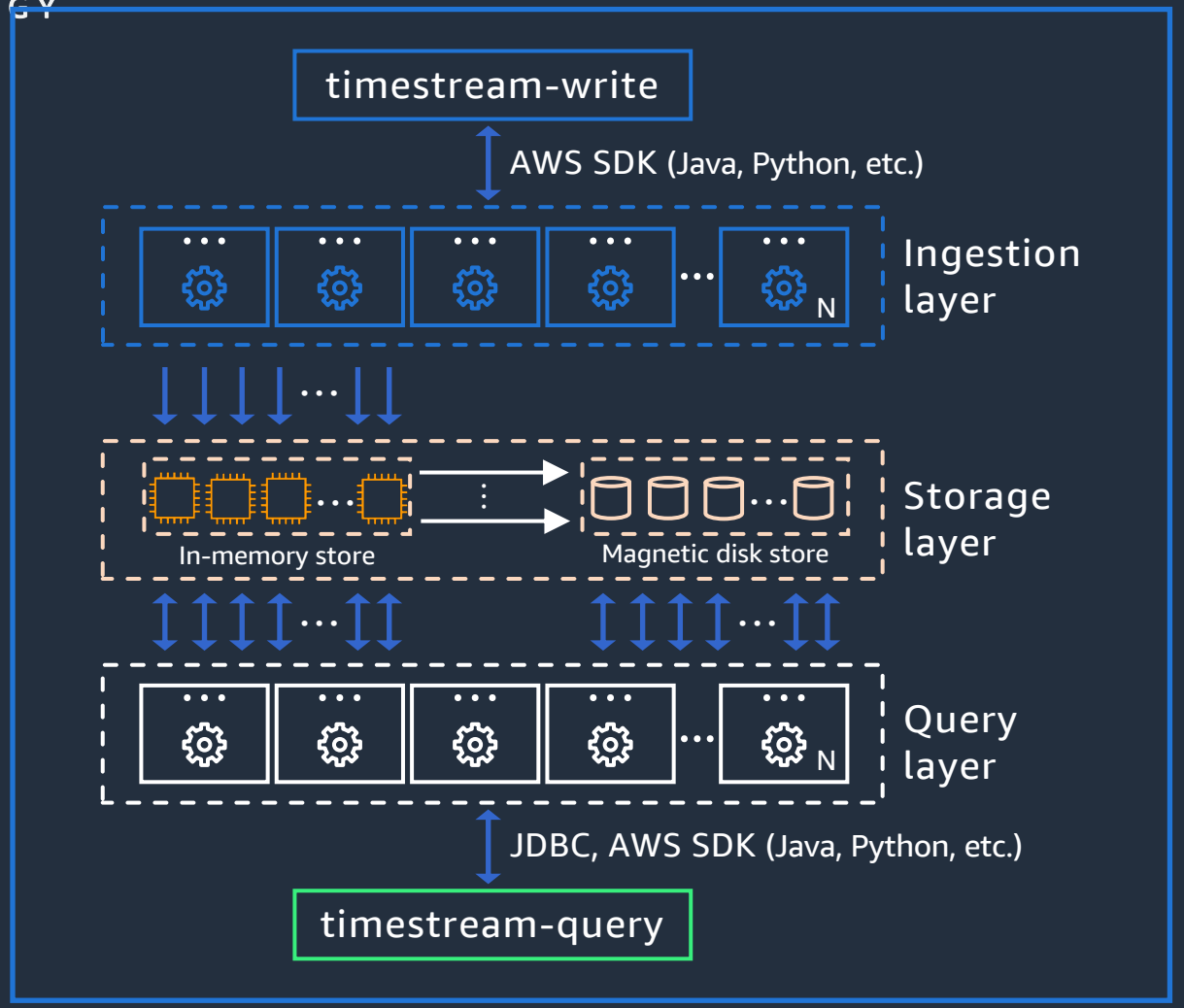
SERVERLESS, CLOUD-NATIVE AWS BUILT TECHNOLOGY

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

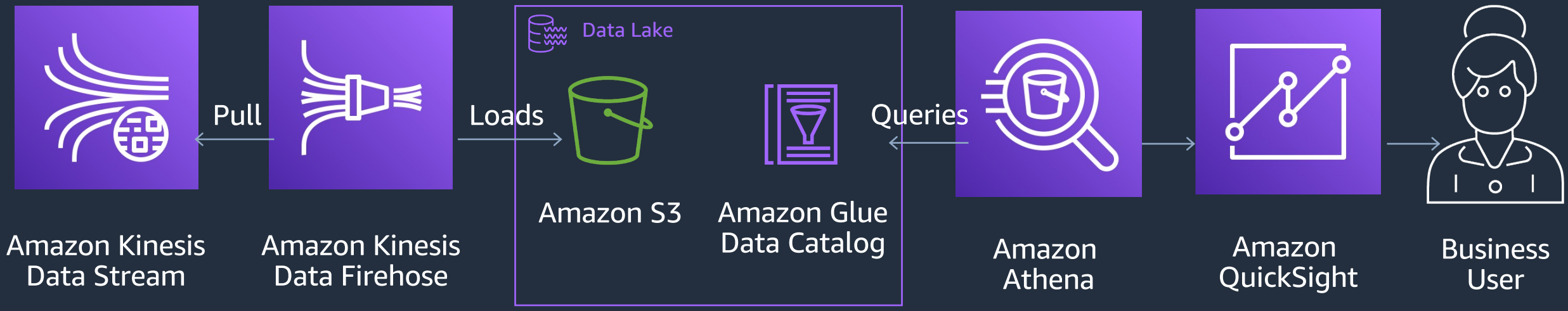


Operational Views

AMAZON MANAGED SERVICE FOR GRAFANA

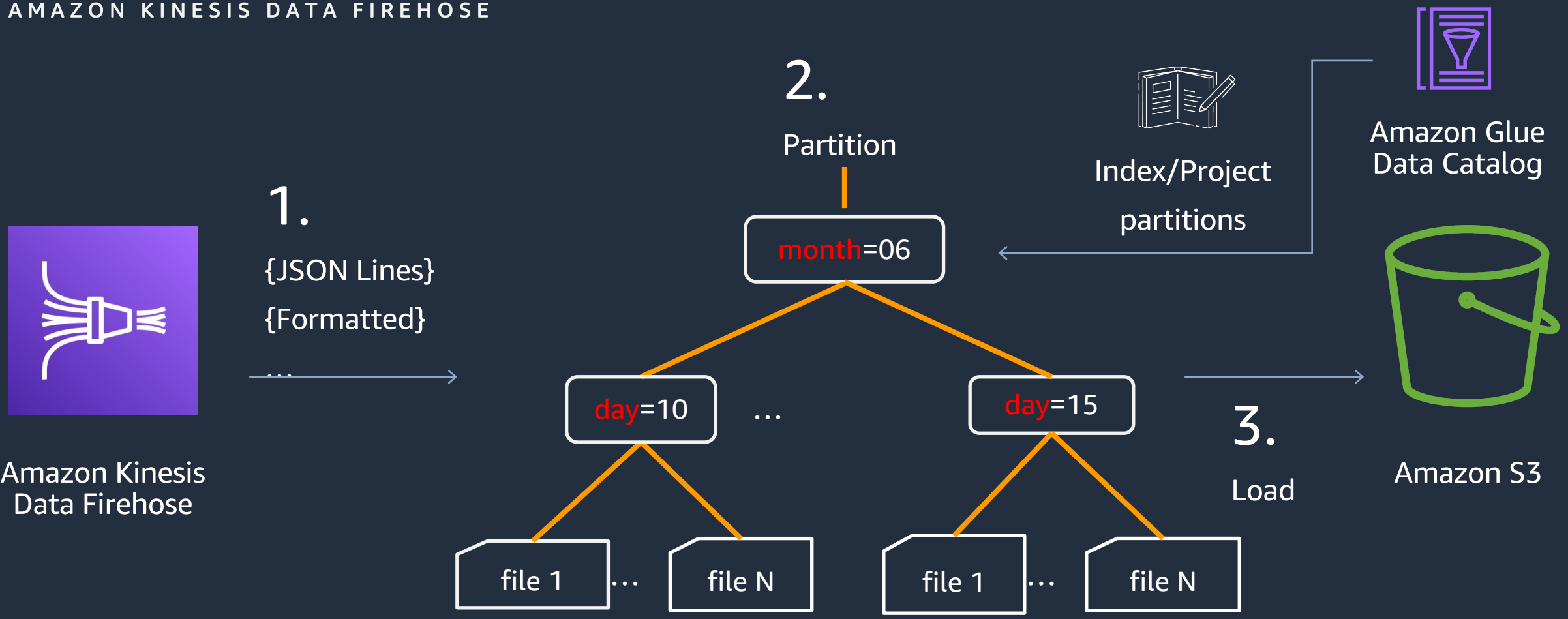
Big data insights

Strategic Big Data Insights Architecture



Data Encoding and Partitioning

AMAZON KINESIS DATA FIREHOSE



Data Encoding and Partitioning

AMA

```
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

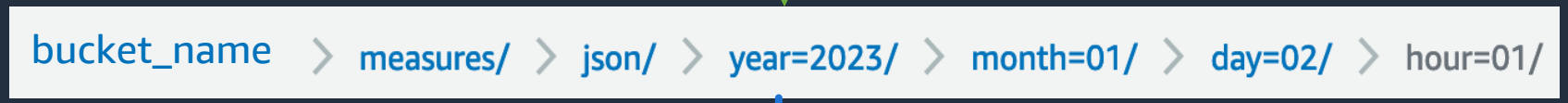
Late arriving data compensation

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}/"
```

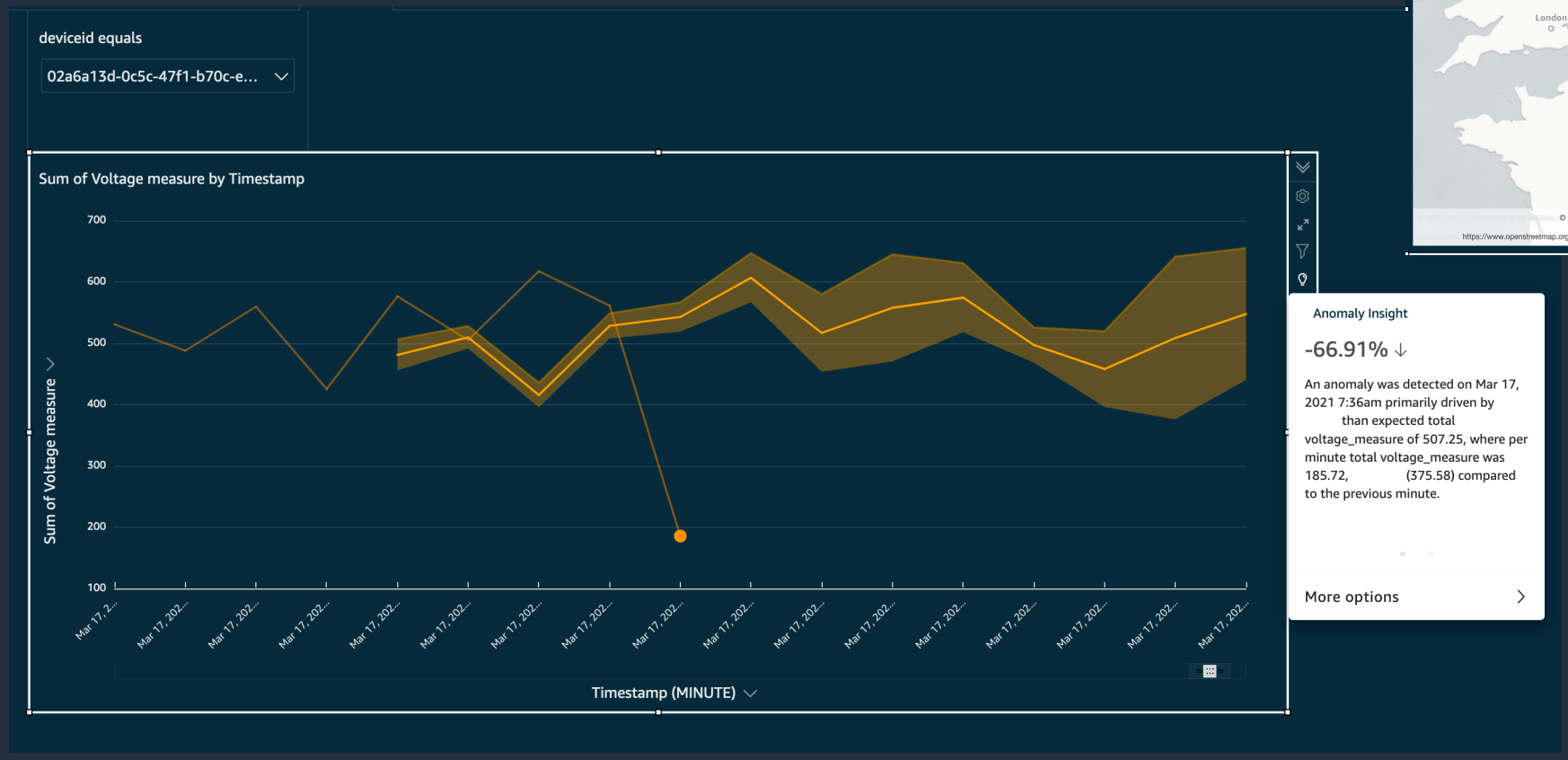


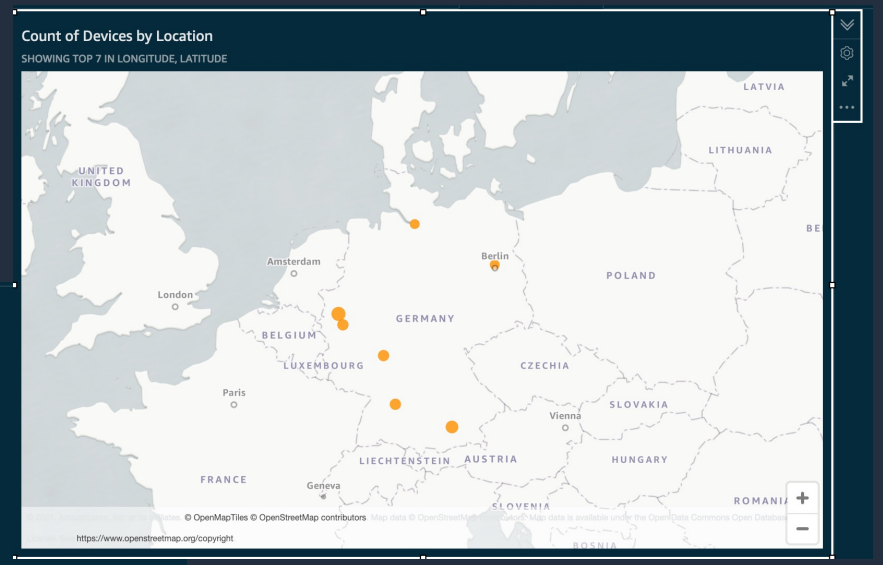

```
CREATE EXTERNAL TABLE raw_device_data (
 ...
)
...
PARTITIONED BY (
 year  INT,
 month INT,
 day   INT,
 hour  INT,
)
LOCATION "s3://DOC-EXAMPLE-BUCKET/prefix/"
TBLPROPERTIES (
 "projection.enabled" = "true",
 "projection.year.type" = "integer",
 "projection.year.range" = "2011,2099",
 "projection.year.digits"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

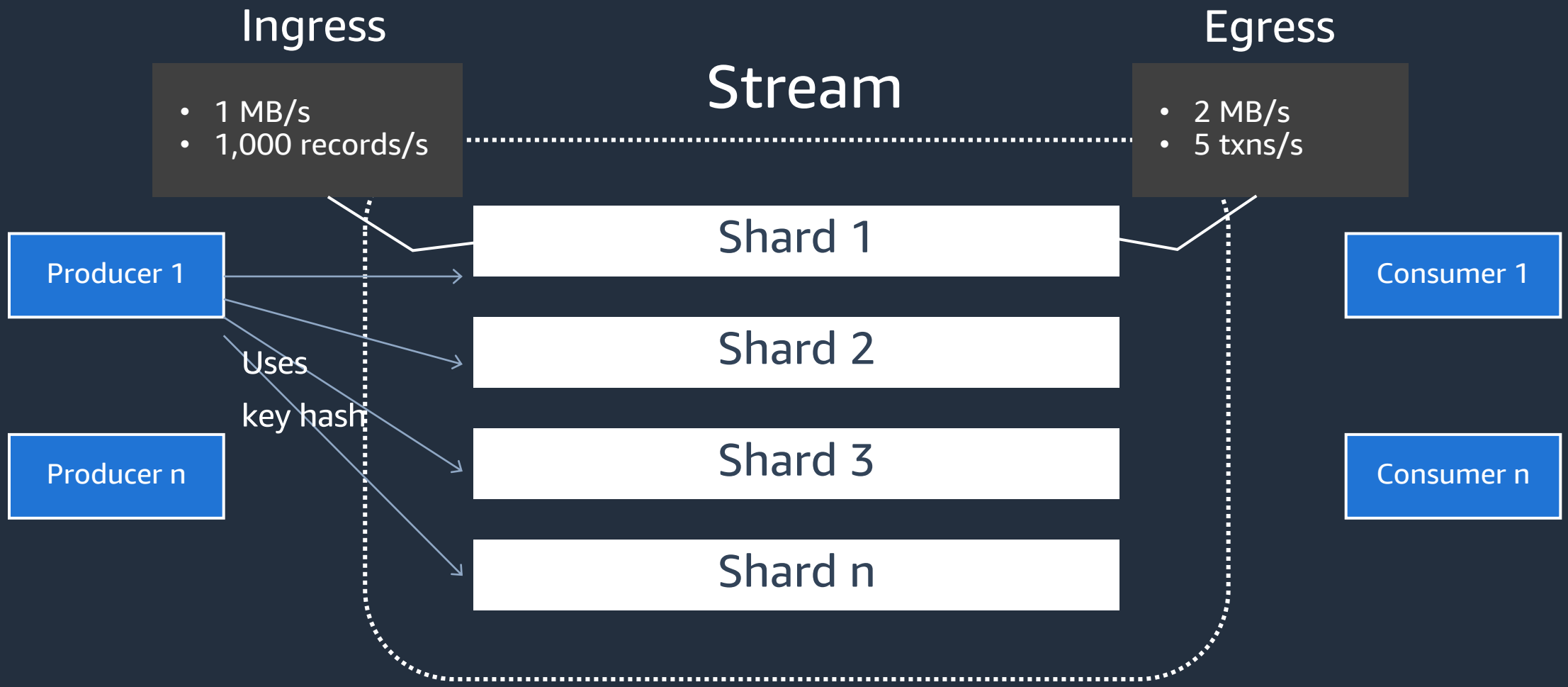
AMAZON QUICKSIGHT



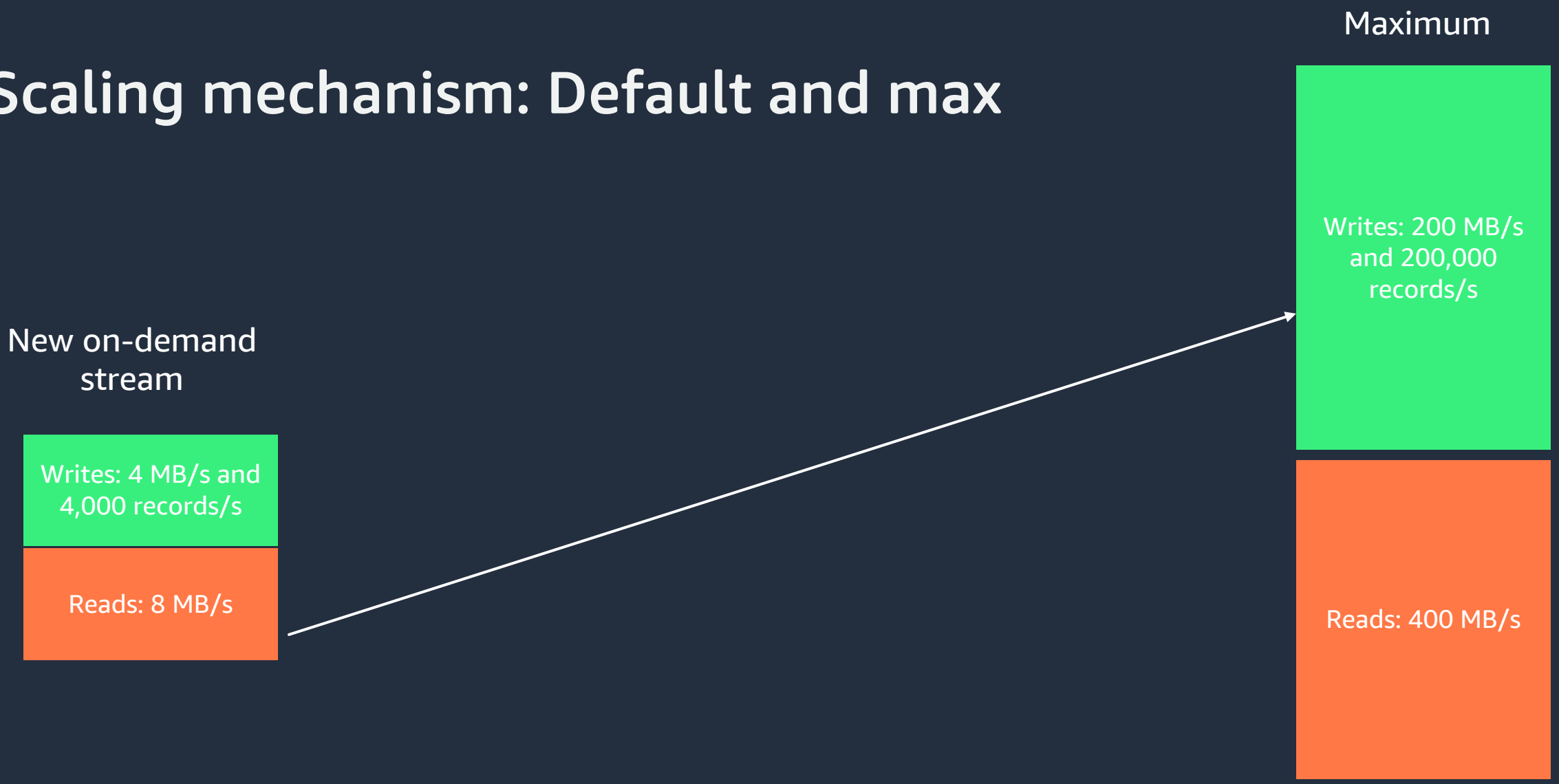


Data stream

Kinesis Data Streams: Granular Provisioning and Scaling

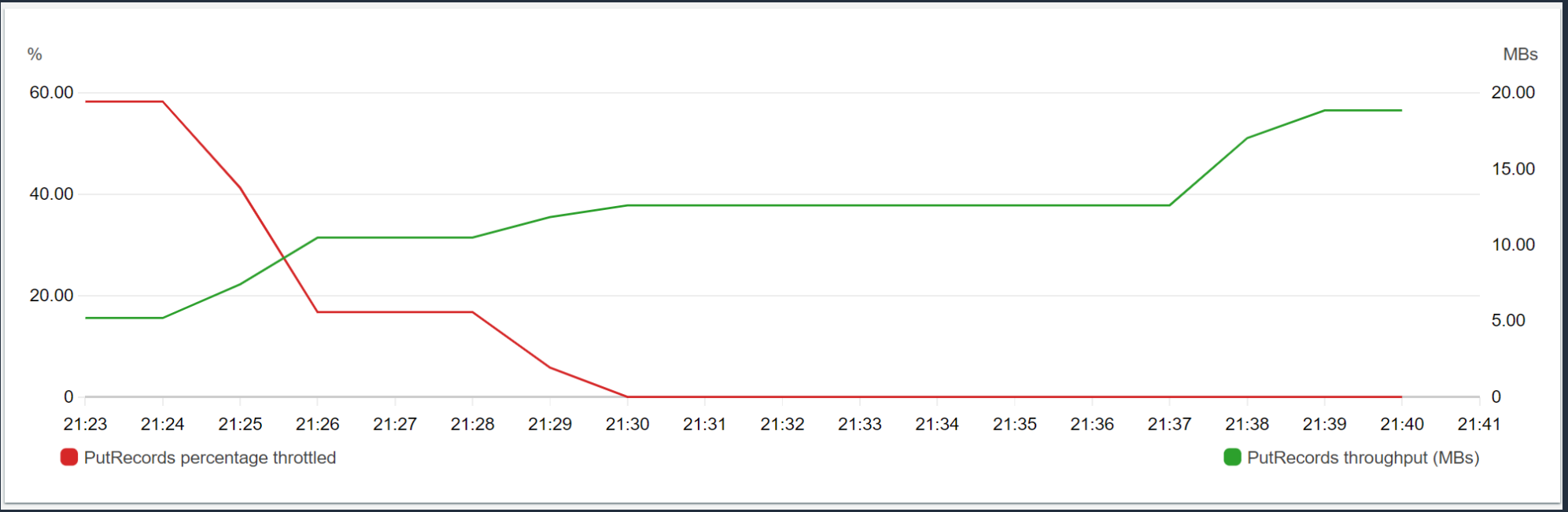


Scaling mechanism: Default and max



Scaling mechanism in action

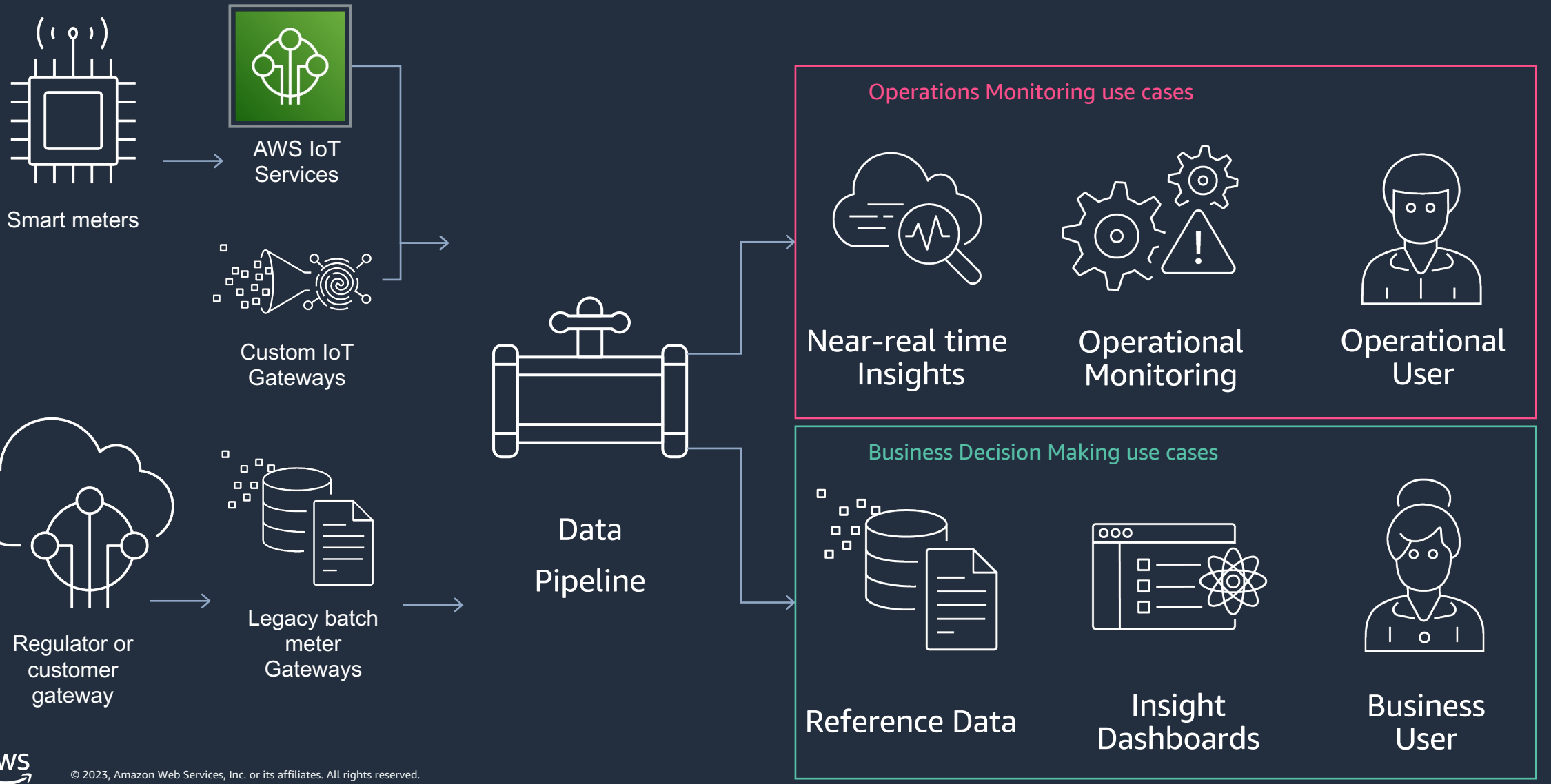
- At time stamp 21:23, we change the stream capacity from provisioned to on-demand
- The stream immediately scales to 10 shards and throttling is reduced
- Now the traffic continues to increase to 18 MB/s, so KDS scales to 40 shards by 21:30 and there are no throttles
- You can also see that the throttles start to reduce within minutes after moving the stream into on-demand and are fully eliminated even as traffic increases

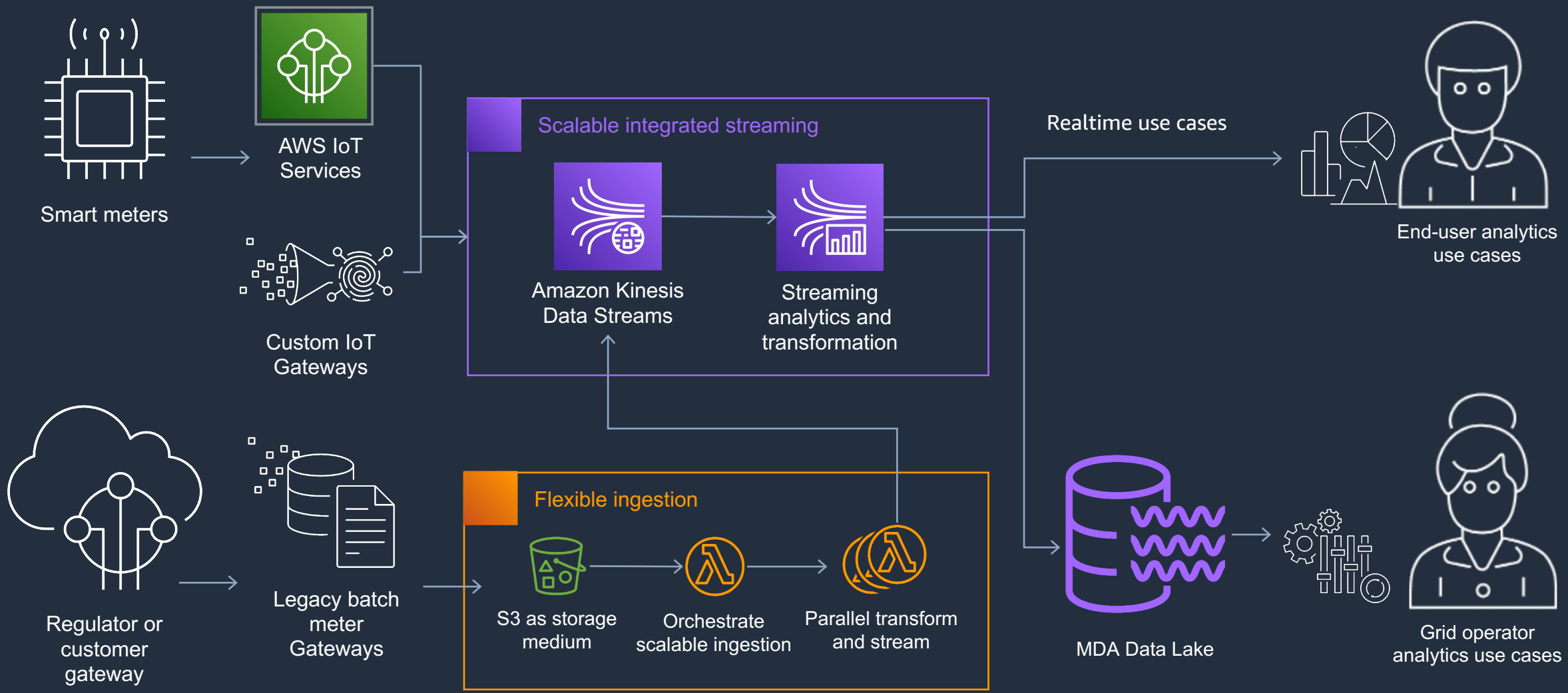


Ingestion

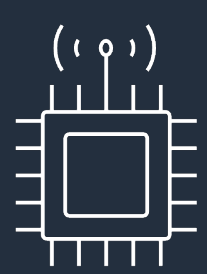
Your stream should not scale to all kinds of ingestion

STREAMING ANALYTICS FOR TIME-SERIES DATA





STREAMING ANALYTICS FOR TIME-SERIES DATA



Smart meters

AWS IoT Services

Custom IoT Gateways

Regulator or customer gateway

Legacy batch meter Gateways

Producer aggregates

Measure A, device 1
{v
{v

Measure B, device 1
{v1, t1}
{v2, t2}

{Device 1, measure A, v1, t1}
{Device 1, measure A, v2, t2}
{Device 1, measure B, v1, t1}
{Device 1, measure B, v2, t2}

{Device 1, {measure A, v1}, {measure B, v1}, t1}
{Device 1, {measure A, v2}, {measure B, v2}, t2}

{Device 1, {measure A, v1}, {measure B, v1}, t1}{Device 1, {measure A, v2}, {measure B, v2}, t2}

Amazon Kinesis Data Streams

aws

```
agg = SimpleJsonRecordAggregator()
agg.on_record_complete(callback=consume_staging_records_intake, executor= map_async)

response = s3_select(bucket, prefix, "SELECT * FROM S3Object", start_range, end_range)
"""
    select_object_content() response is an event stream that can be looped to concatenate the overall result set
    Hence, we are joining the results of the stream in a string before converting it to a tuple of dict
"""
for index, payload in enumerate(response["Payload"]):
    if records := payload.get("Records"):
        staging_records = (records["Payload"].decode("utf-8")).split("\n")
        for some_staging_record in staging_records:
            agg.add_record(some_staging_record["meter_id"], some_staging_record)
agg.flush()
```

Reactively producer to stream

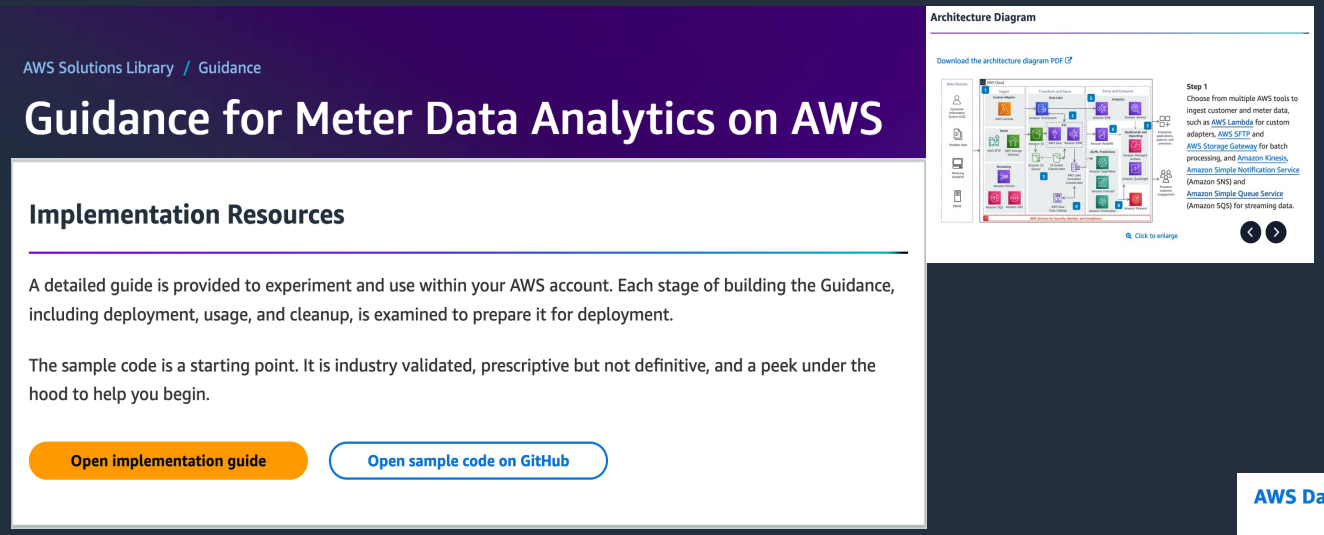
Stream from S3

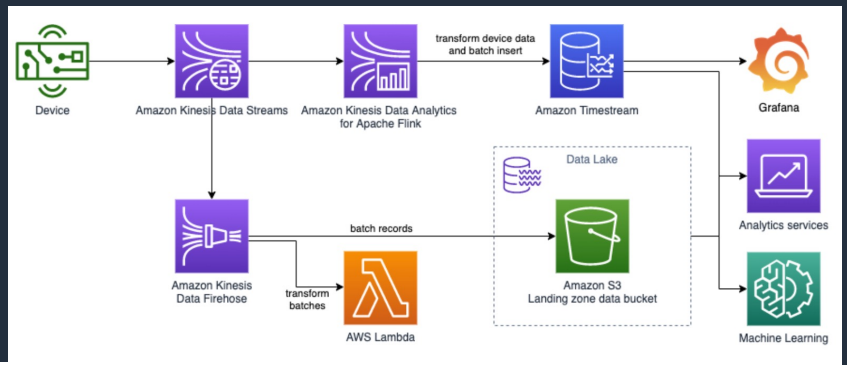
Add records to aggregator

Summary

Getting started

https://aws.amazon.com/solutions/guidance/meter-data-analytics-on-aws/





AWS Database Blog

Near real-time processing with Amazon Kinesis, Amazon Timestream, and Grafana

by John Mousa and Sascha Janssen | on 12 JUL 2021 | in Advanced (300), Amazon Kinesis, Amazon Managed Service For Apache Flink, Amazon Timestream, Technical How-To | Permalink | Comments | Share

https://github.com/aws-samples/amazon-kinesis-timestream-grafana

https://www.pulse.aws/survey/FL1FAATO

Thank you!

John Mousa

Direct link to slides right after 5-click survey feedback

“

”

Thank you!