AWS re:Invent

SVS323-R

Mastering AWS Lambda streaming event sources

Adam Wagner
Solutions Architect
Amazon Web Services

AWS re:Invent

aws

Related breakouts

SVS317-R – Serverless stream processing pipeline best practices

SVS401-R – Optimizing your serverless applications

SVS335-R – Serverless at scale: Design patterns and optimizations

API304 – Scalable serverless event-driven applications using Amazon SQS & Lambda

Agenda

Introduction to streaming event sources for AWS Lambda

Scaling

Monitoring and error handling

Common issues

Performance and optimization

Session expectations

- Chalk-talk format – Please ask questions
- What we will cover
  - The details of using Lambda with streaming event sources
    - Scaling
    - Monitoring
    - Error handling
    - Performance and optimization
- What we won’t cover
  - What is serverless?
  - What is Lambda?
  - Event sources outside of Amazon Kinesis Data Streams and Amazon DynamoDB Streams

Streaming event sources

Amazon Kinesis

Easily collect, process, and analyze video and data streams in real time

Amazon Kinesis Video Streams

Capture, process, and store video streams for analytics

Amazon Kinesis Data Streams

Build custom applications that analyze data streams

Amazon Kinesis Data Firehose

Load data streams into AWS data stores

Amazon Kinesis Data Analytics

Analyze data streams with SQL

We will focus on Kinesis Data Streams

Amazon DynamoDB

Amazon DynamoDB

Fully managed NoSQL

Document or key-value

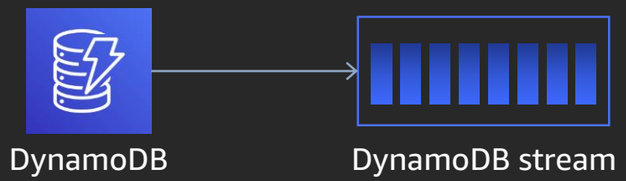
Scales to any workload

Fast and consistent

Access control

Event-driven programming

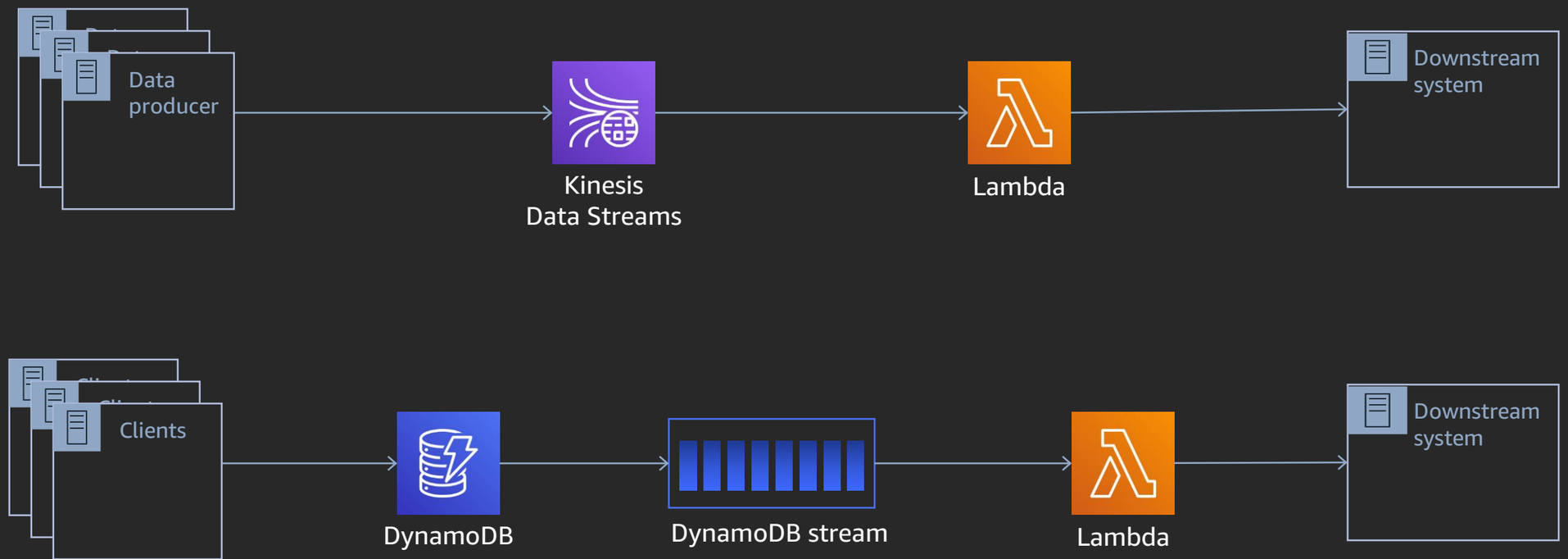
DynamoDB Streams



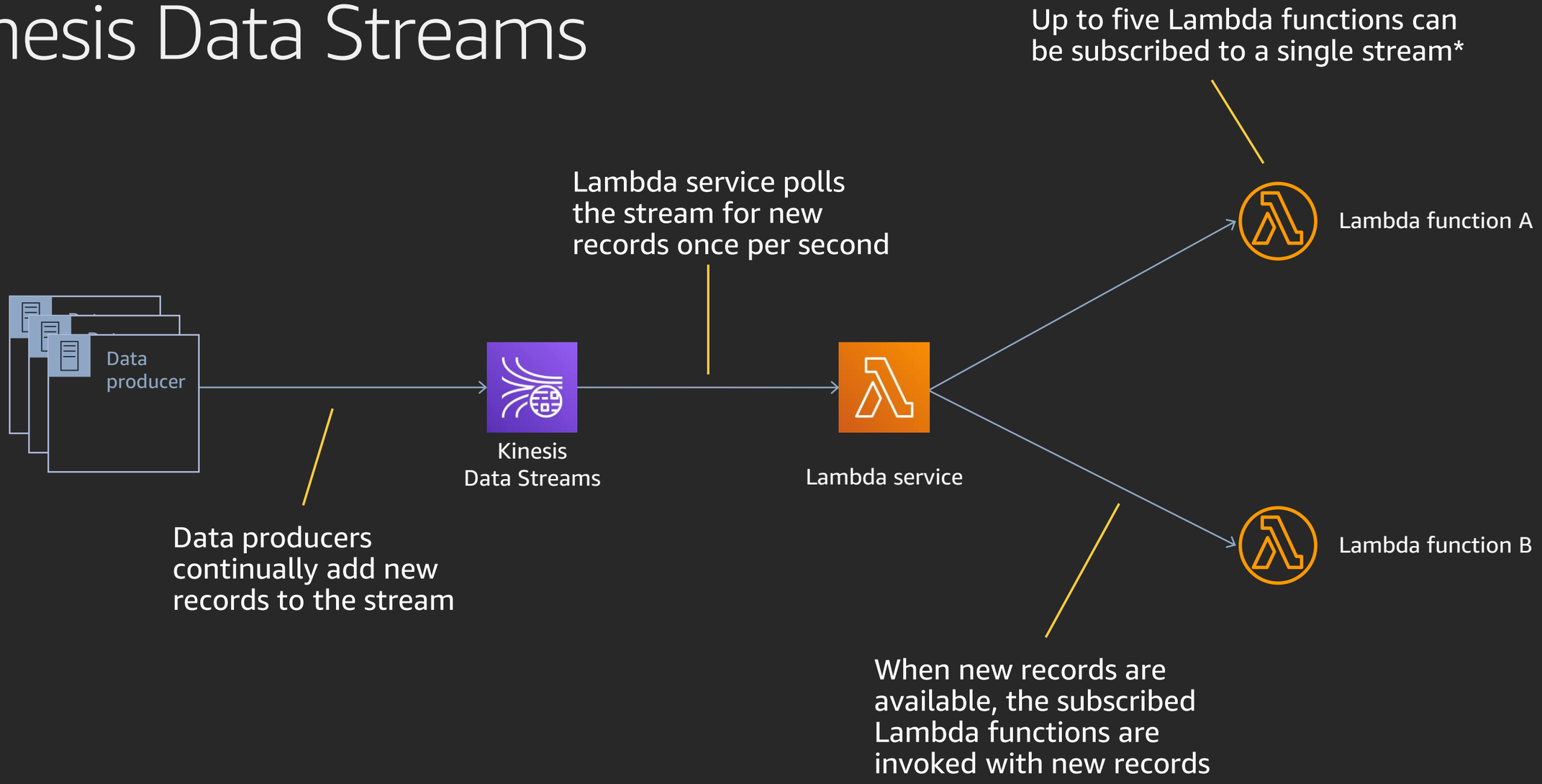
DynamoDB Streams

- ✓ Stream of item changes
- ✓ Exactly once, guaranteed delivery
- ✓ Strictly ordered by key
- ✓ Durable, scalable
- ✓ Fully managed
- ✓ 24-hour data retention
- ✓ Sub-second latency
- ✓ Event source for Lambda

What we're talking about today

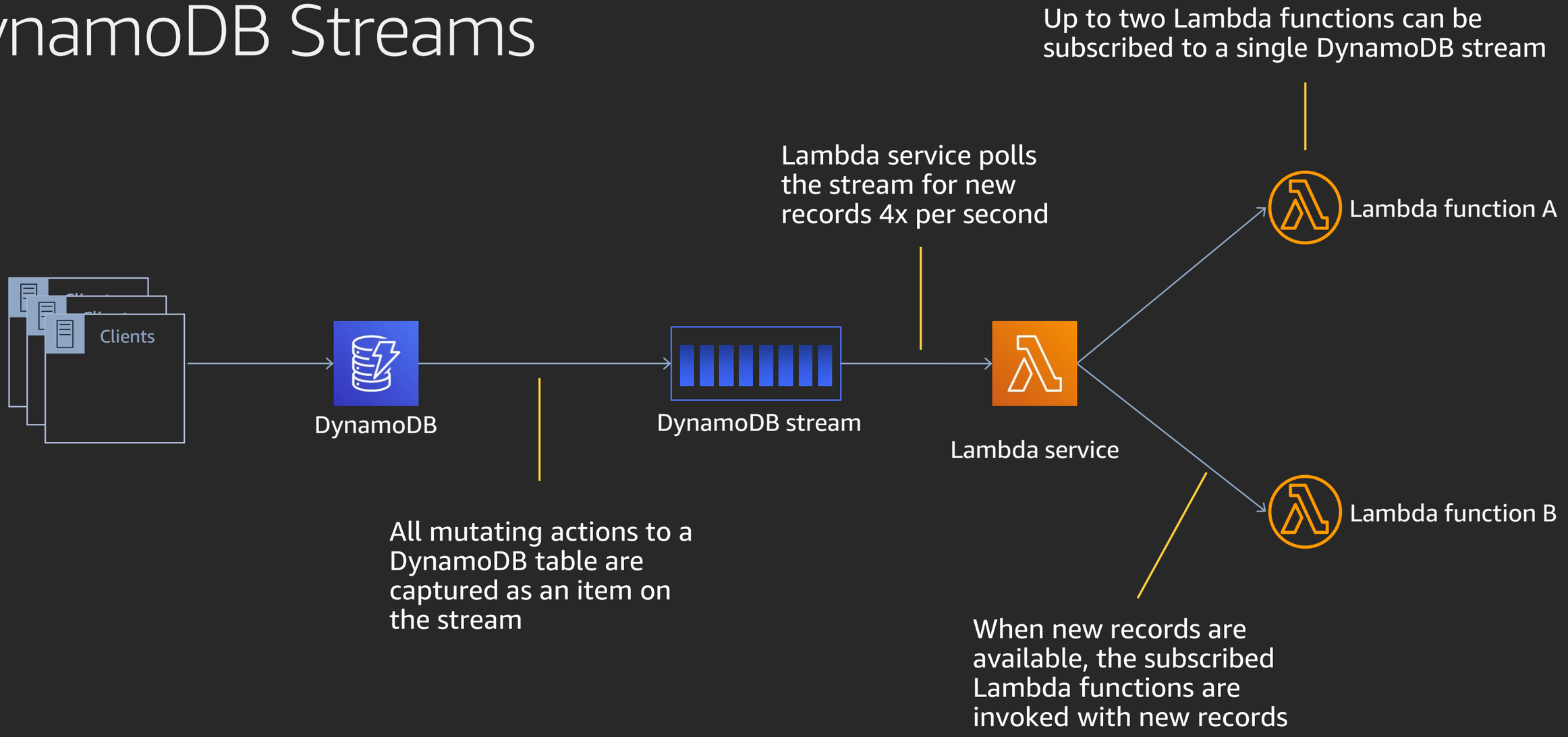


Kinesis Data Streams

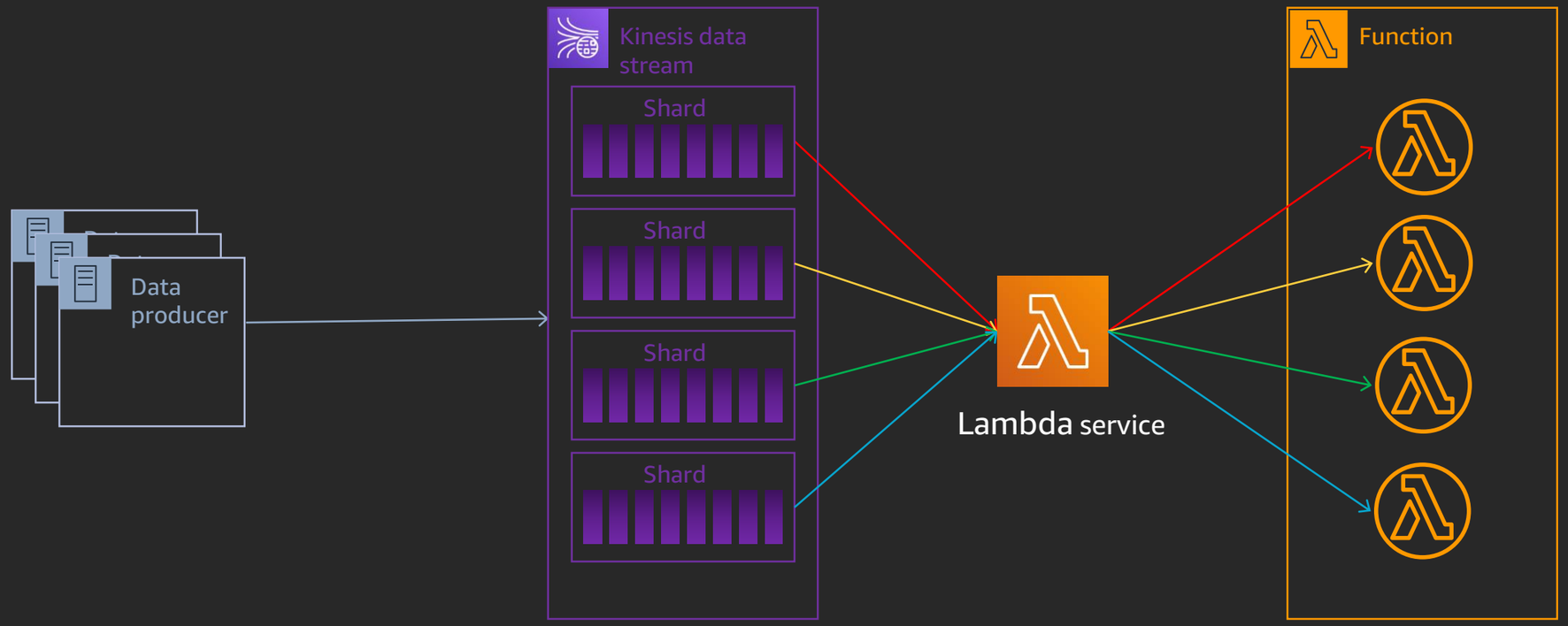


*Kinesis Data Analytics and Kinesis Data Firehose also count toward this subscriber limit

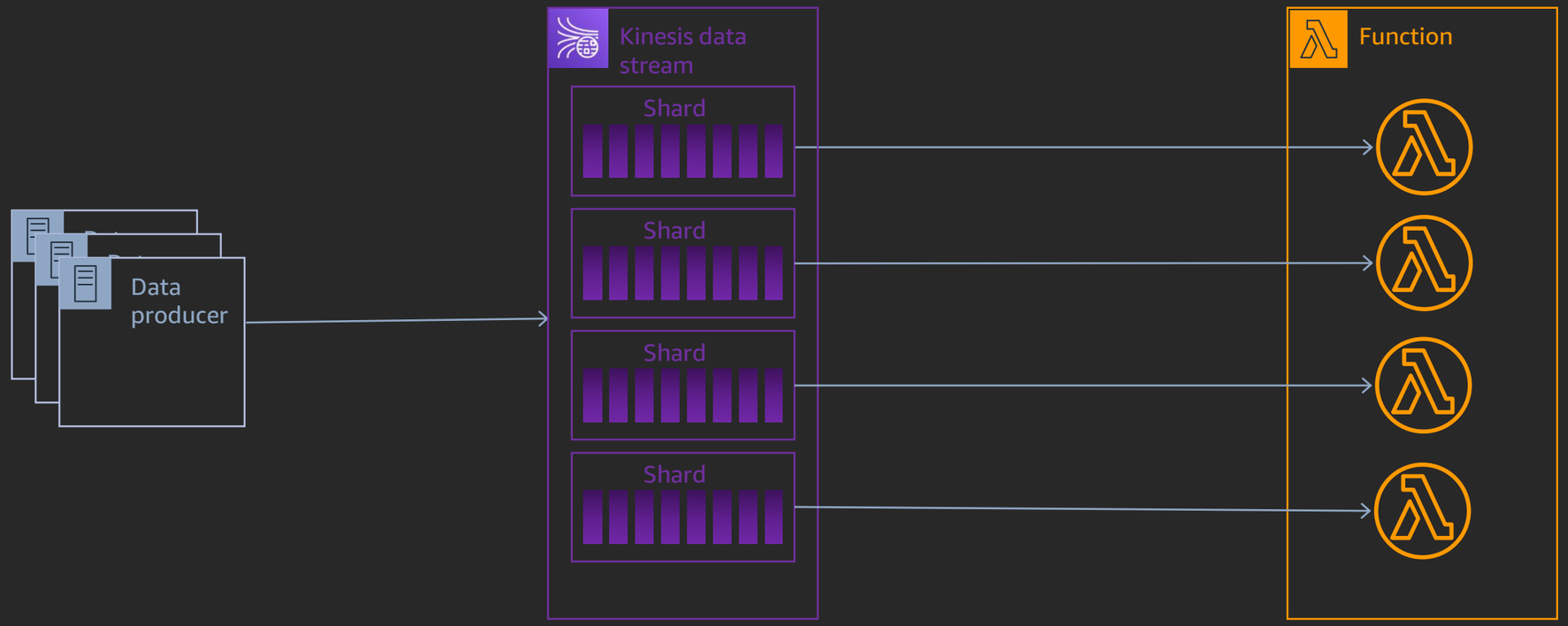
DynamoDB Streams



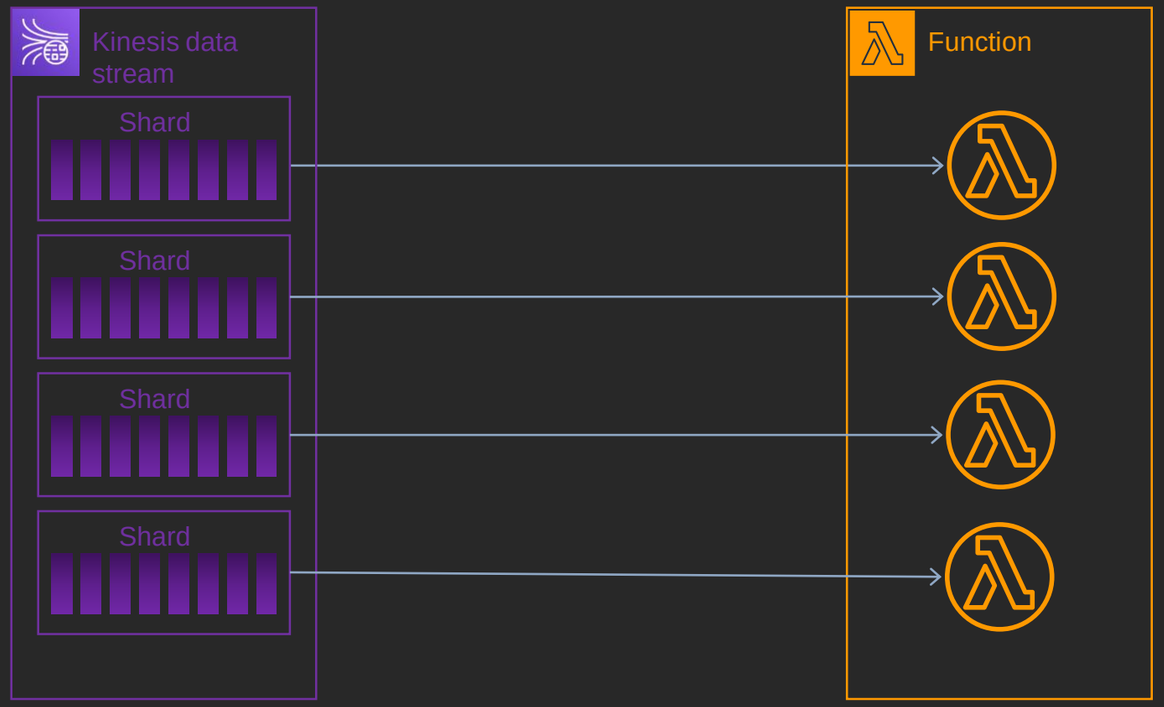
Kinesis data stream shard detail



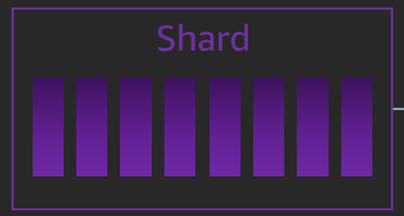
Kinesis data stream



Kinesis data stream



Kinesis data stream shard-level detail



1. Lambda service polls the shard once per second for a set of records. Then synchronously invokes the Lambda function with the batch of records.

2. If the Lambda returns successfully, the Lambda service advances to the next set of records and repeats step 1.

3. If the Lambda errors, by default the Lambda service invokes the function with the same set of records and will continue to do so until it succeeds or the records age out of the stream.

Scaling

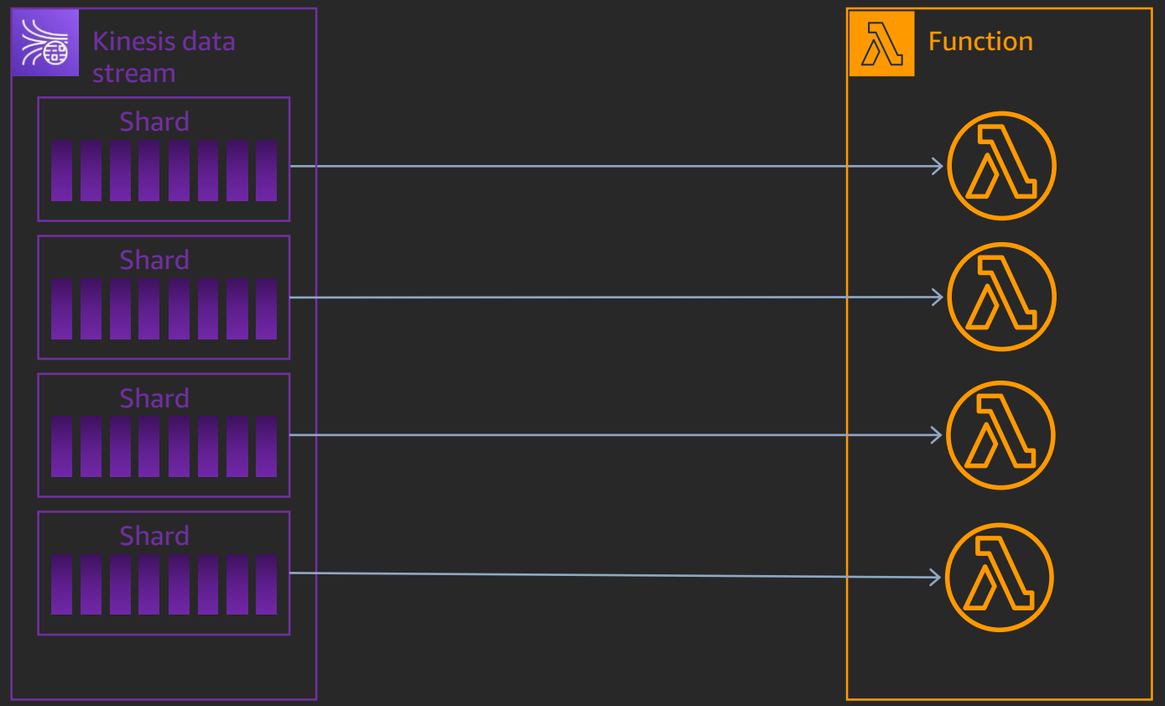
Kinesis data stream scaling

Kinesis Data Streams scales by adding shards

Easiest method is the UpdateShardCount API

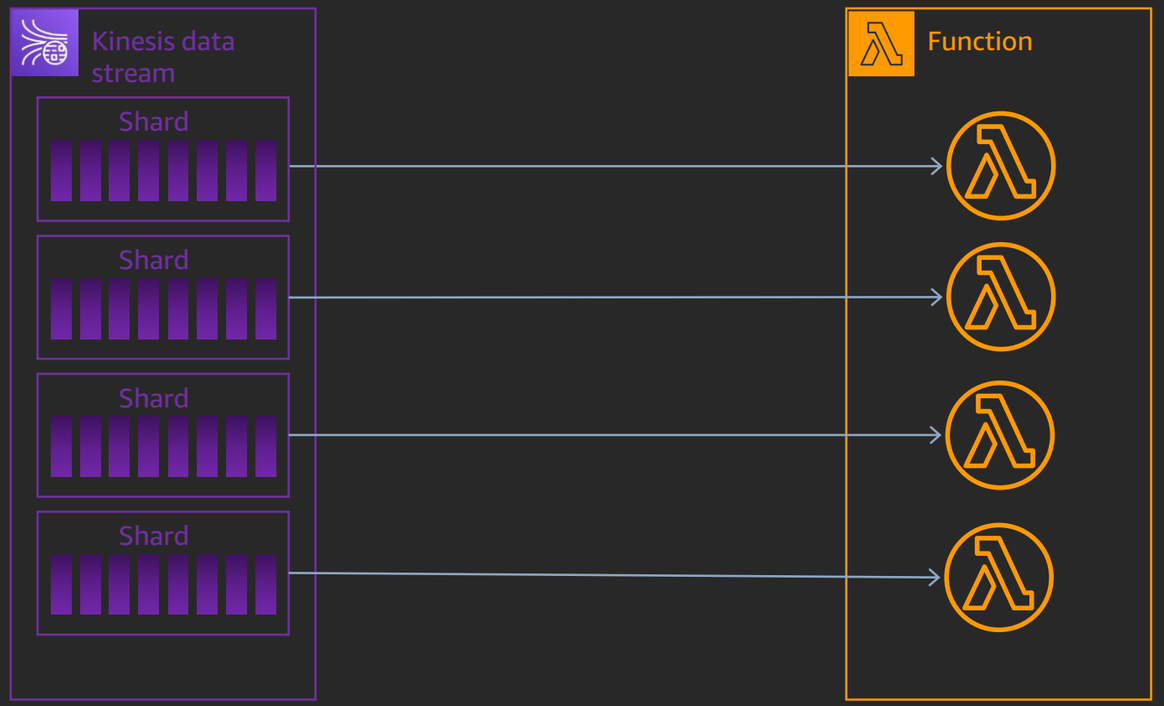
- Set a target shard count and Kinesis Data Streams takes care of splitting and merging shards for you

Alternatively, SplitShard and MergeShards allow for more targeted scaling



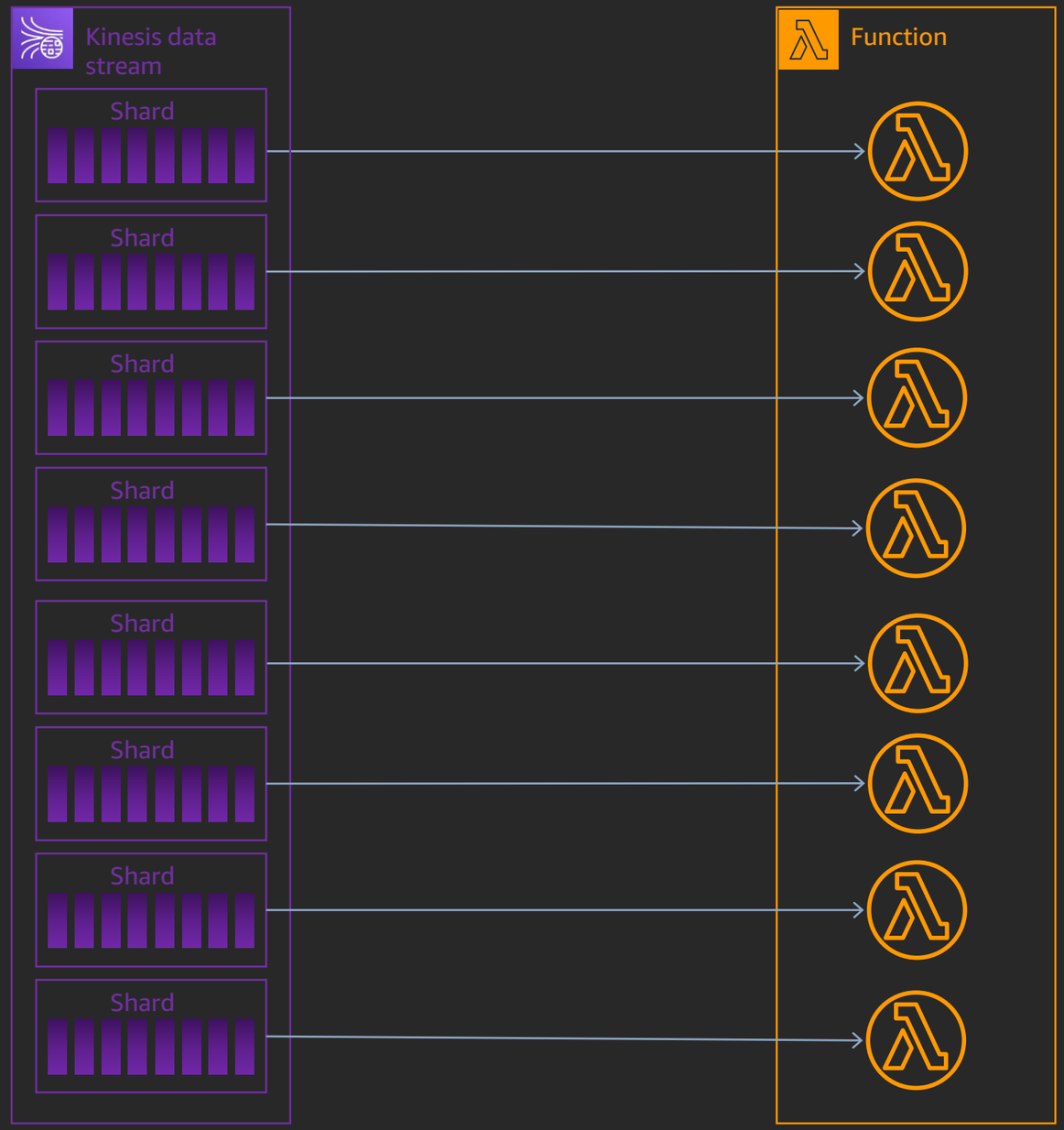
Kinesis data stream scaling

```
aws kinesis update-shard-count --stream-name reinvent19-01
--target-shard-count 8 --scaling-type UNIFORM_SCALING
{
    "StreamName": "reinvent19-01",
    "CurrentShardCount": 4,
    "TargetShardCount": 8
}
```



Kinesis data stream scaling

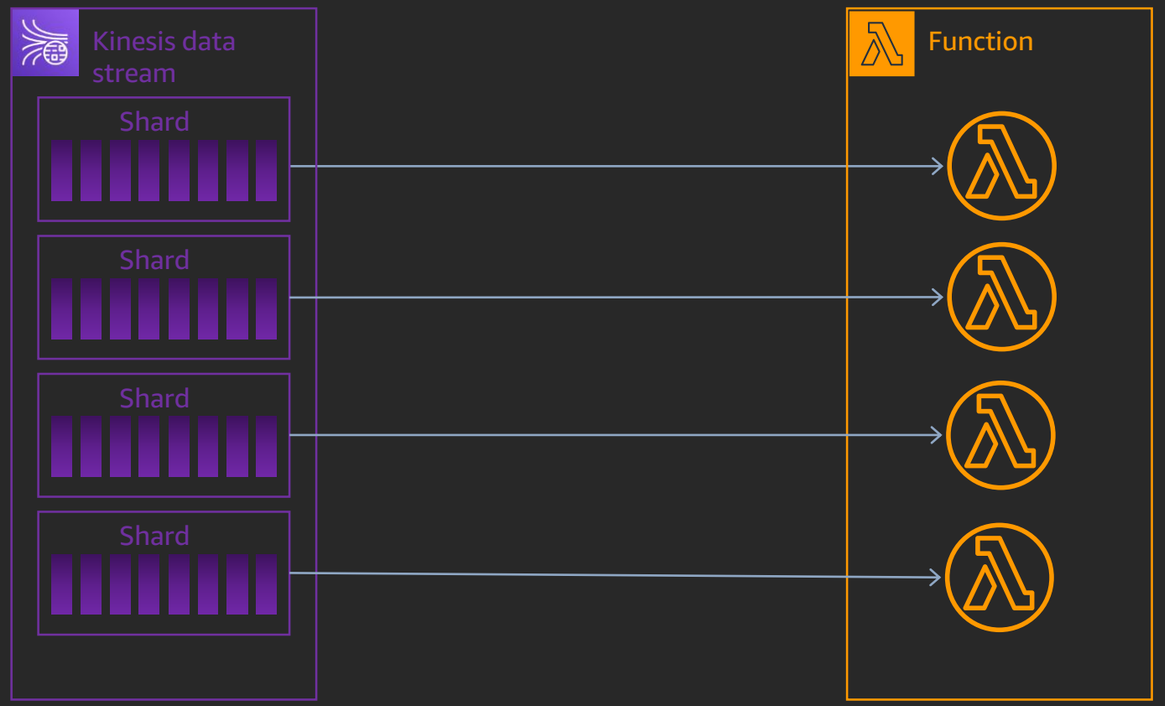
```
aws kinesis update-shard-count --stream-name reinvent19-01
--target-shard-count 8 --scaling-type UNIFORM_SCALING
{
    "StreamName": "reinvent19-01",
    "CurrentShardCount": 4,
    "TargetShardCount": 8
}
```



Kinesis data stream scaling

UpdateShardCount API limits

- Scale more than twice per rolling 24-hour period per stream
- Scale up to more than double your current shard count for a stream
- Scale down below half your current shard count for a stream
- Scale up to more than 500 shards in a stream
- Scale a stream with more than 500 shards down unless the result is less than 500 shards
- Scale up to more than the shard limit for your account

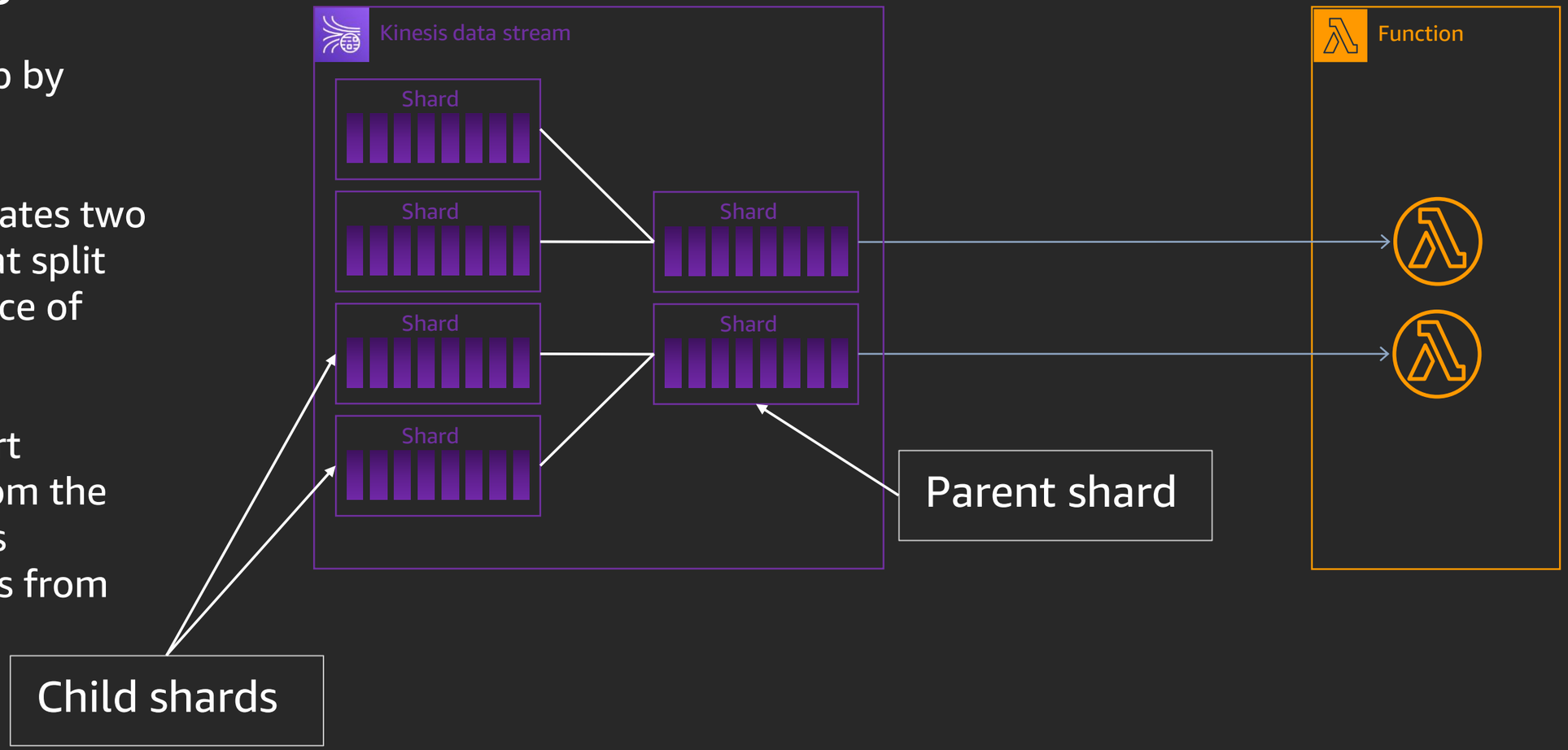


Most of these limits can be adjusted through AWS Support

Kinesis data stream scaling … more detail

Splitting shards

- The stream scales up by splitting shards
- Splitting a shard creates two new child shards that split the partition keyspace of the parent shard
- Lambda will not start receiving records from the child shards until it's processed all records from the parent shard



Throughput considerations

Kinesis max message ingest per shard

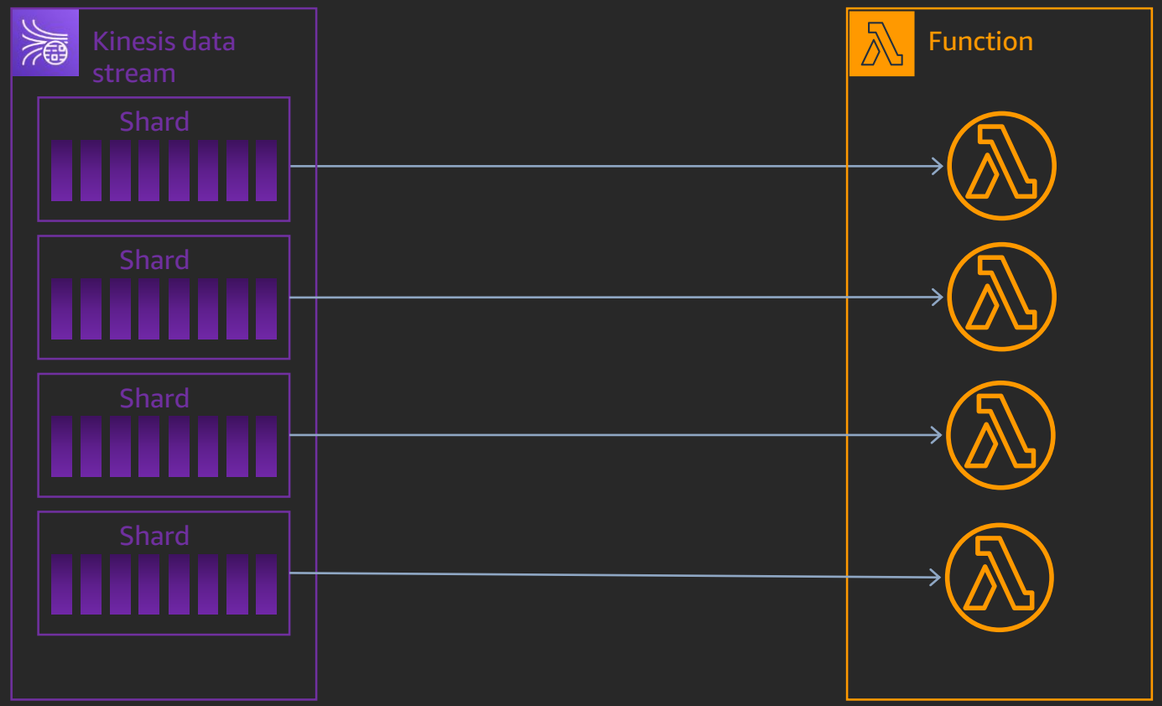
MIN (1 MiB/average record size, 1,000)

Lambda maximum messages: Steady state

MIN (1 MiB/average record size, 1,000)

Lambda maximum messages: Edge cases

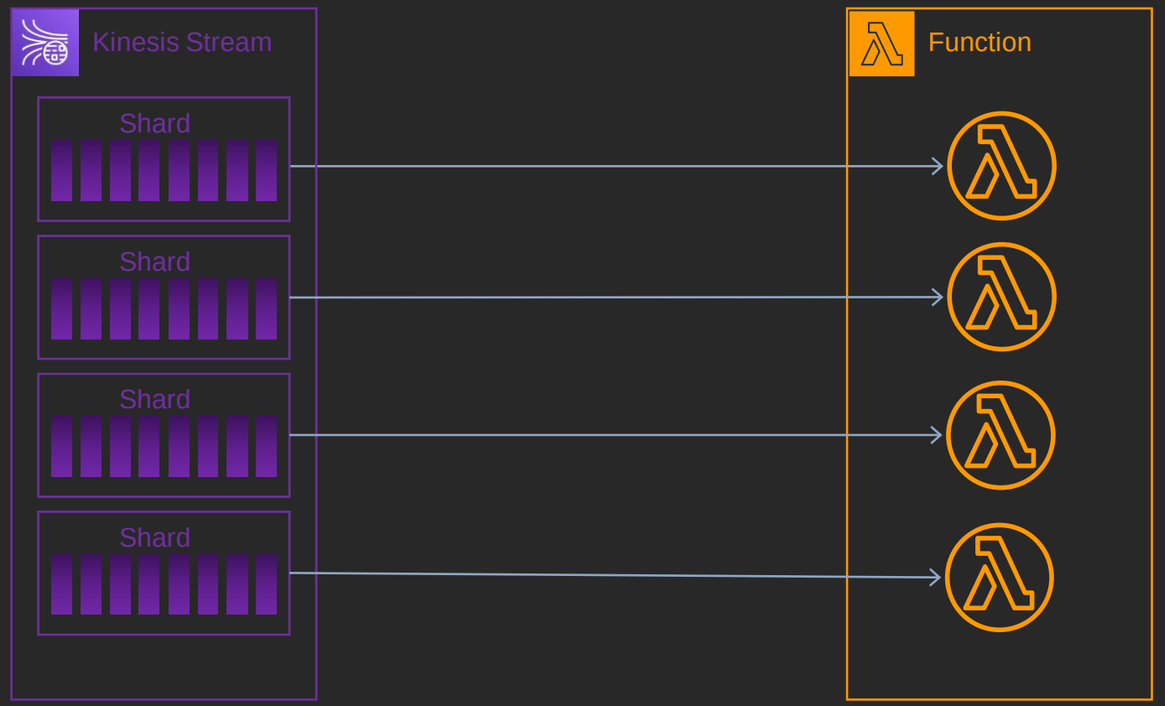
MIN (6 MiB/average record size, 10,000)



Parallelization Factor

NEW

- Adds Lambda parallelization per shard
- Setting of 1 is the same as the current behavior, maximum setting is 10
- Batching via partition keys to maintain in order processing per partition key
- Works with both Kinesis Data Streams and DynamoDB Streams

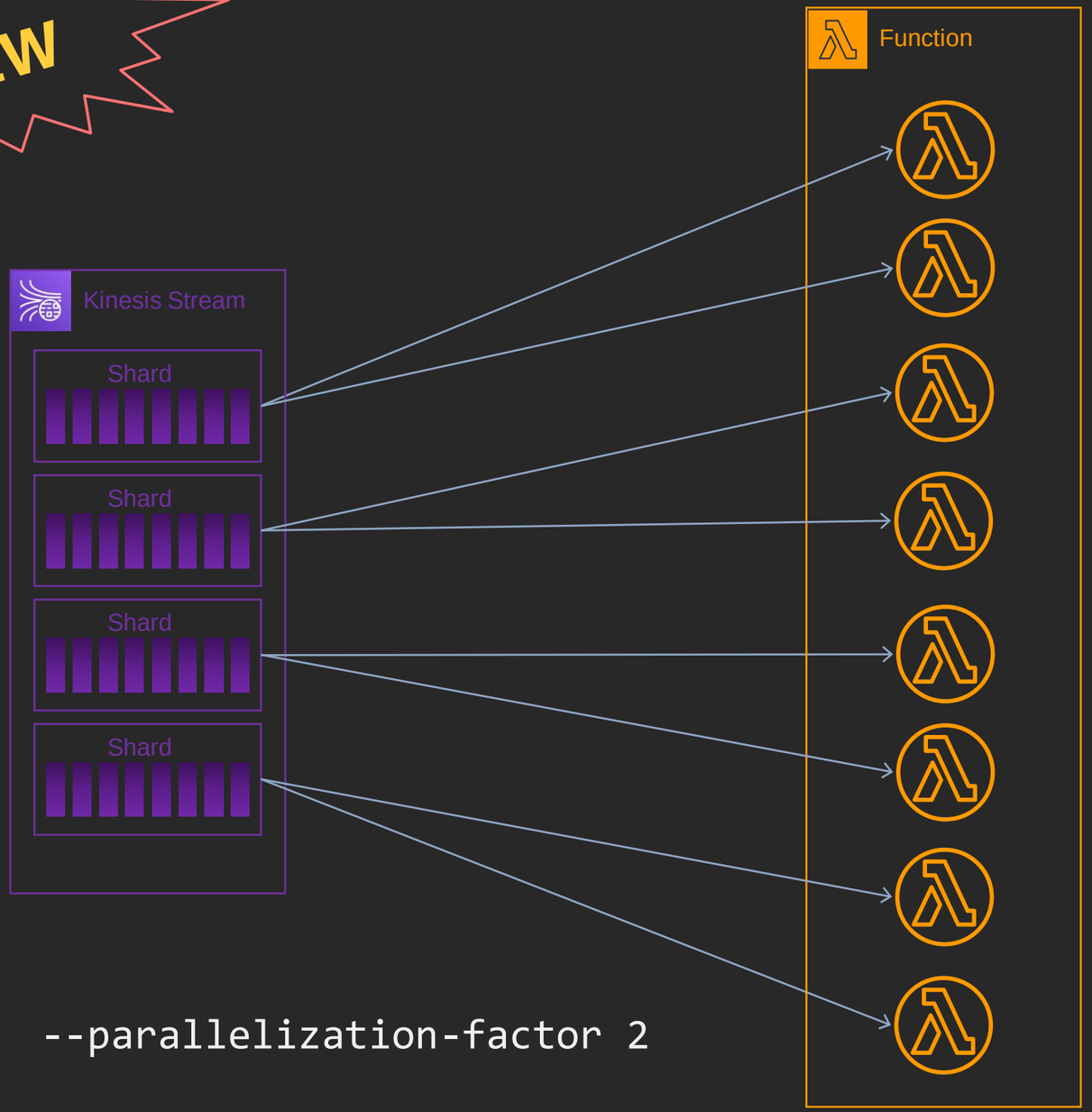



```
--parallelization-factor 1
```

Parallelization Factor

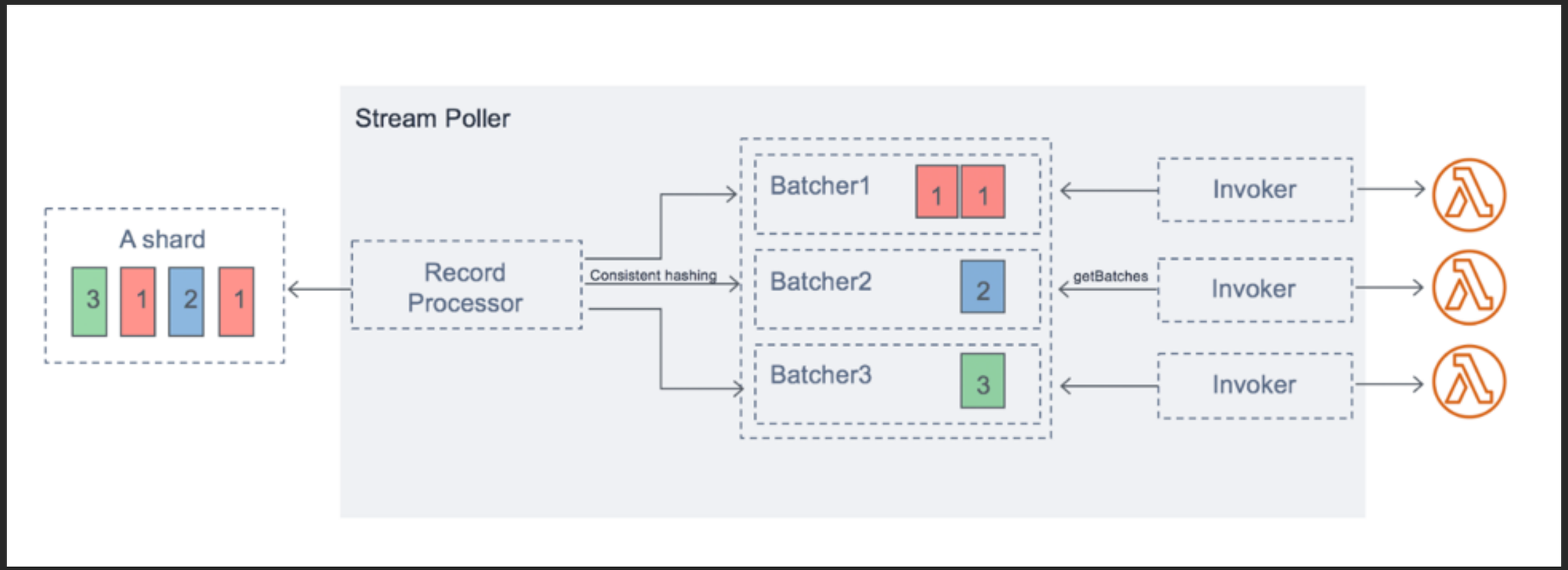
NEW

- Adds Lambda parallelization per shard
- Setting of 1 is the same as the current behavior, maximum setting is 10
- Batching via partition keys to maintain in order processing per partition key
- Works with both Kinesis Data Streams and DynamoDB Streams



Parallelization Factor

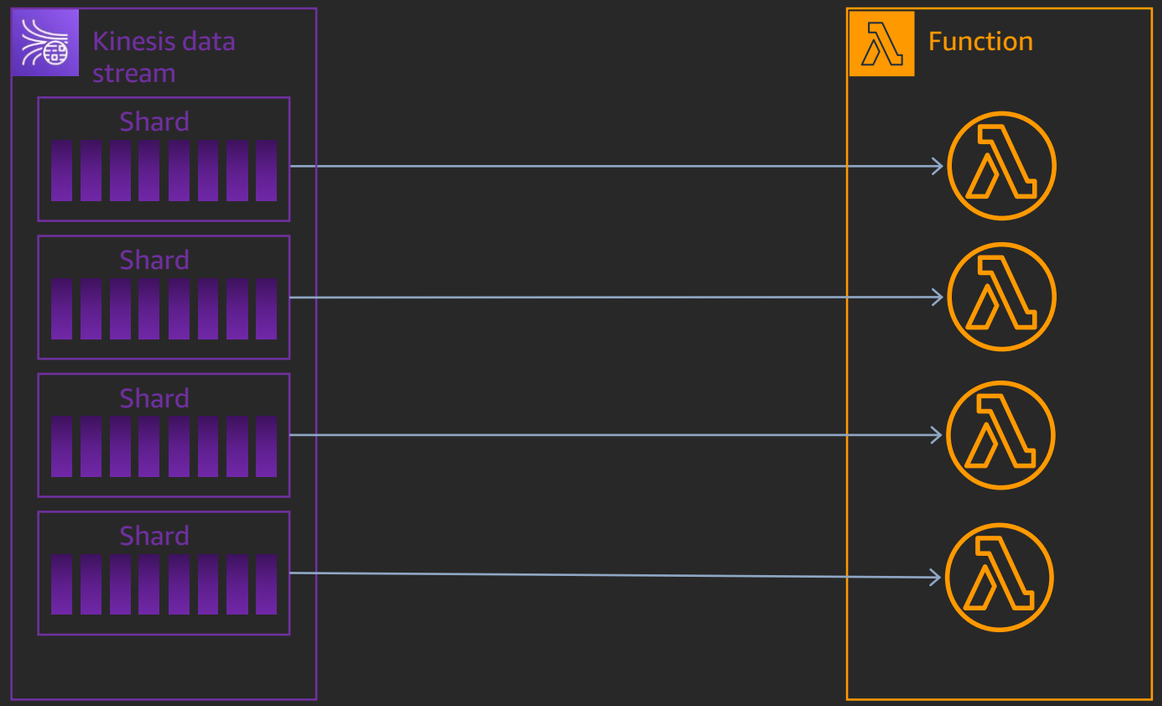
NEW



Kinesis data stream scaling

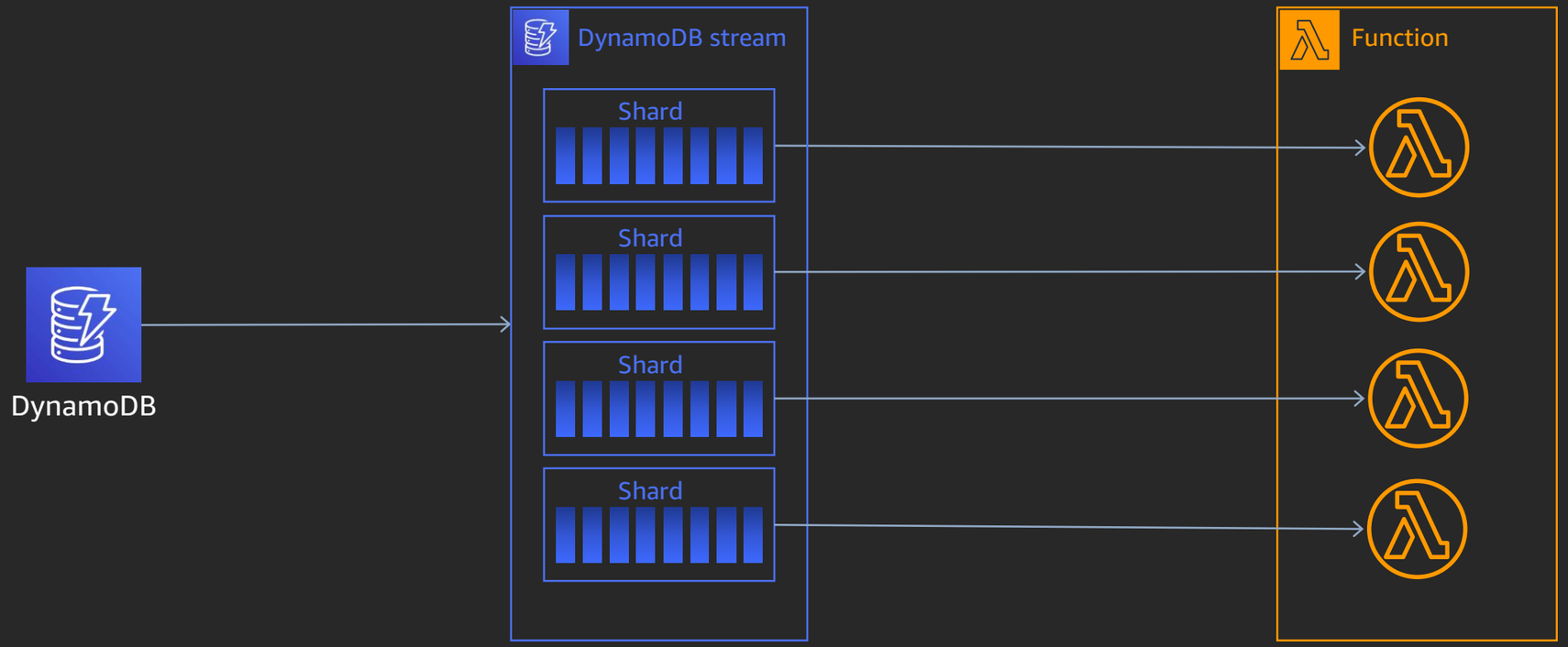
Best practices

- Auto-scale your shard count using Application Auto Scaling: https://aws.amazon.com/blogs/big-data/scaling-amazon-kinesis-data-streams-with-aws-application-auto-scaling/
- Scale conservatively to leave overhead for bursts of traffic
- Scale your shard count to match your Lambda throughput and/or use Parallelization Factor
- Test! Test! Test! Measure unit tests to watch for performance regressions, and also test at scale!



DynamoDB Streams scaling

- Unlike Kinesis, there is no direct control of the number of shards
- The number of shards depends on the volume of data in the table as well as the amount of read and write throughput provisioned for the table



DynamoDB on-demand vs. provisioned capacity

Provisioned capacity

- Set read capacity (RCU) and write capacity (WCU)
- Optionally auto-scale
- Good for steady-state workloads

On-demand

- Pay per request
- Scaling is completely managed
- Good for variable, bursty, workloads
- Matches very well with many serverless workloads

DynamoDB on-demand scaling

DynamoDB on-demand supports large scaling spikes

From the DynamoDB docs:

> DynamoDB tables using on-demand capacity mode automatically adapt to your application's traffic volume. On-demand capacity mode instantly accommodates up to double the previous peak traffic on a table. For example, if your application's traffic pattern varies between 25,000 and 50,000 strongly consistent reads per second where 50,000 reads per second is the previous traffic peak, on-demand capacity mode instantly accommodates sustained traffic of up to 100,000 reads per second. If your application sustains traffic of 100,000 reads per second, that peak becomes your new previous peak, enabling subsequent traffic to reach up to 200,000 reads per second.

In general ,when a table's ability to serve a set amount of traffic doubles, so does the number of shards in the stream

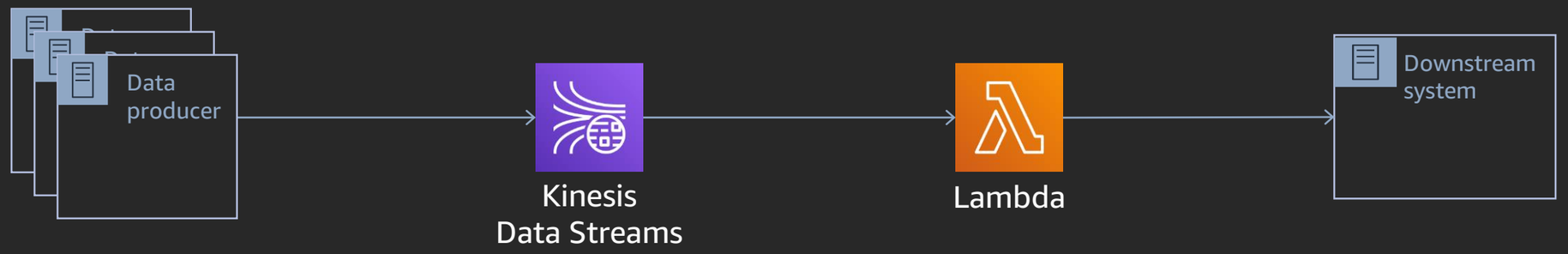
DynamoDB on-demand scaling

Key takeaways

- On-demand mode’s ability to scale quickly is great
- Make sure your Lambda and any downstream services can match that scaling
- Be extra careful if using a Parallelization Factor > 1
    - Monitor your Lambda concurrency
        - Alarm on ConcurrentExecutions at an account level
        - As well as on any functions that have a limit/reservation

Monitoring and error handling

Kinesis data stream monitoring



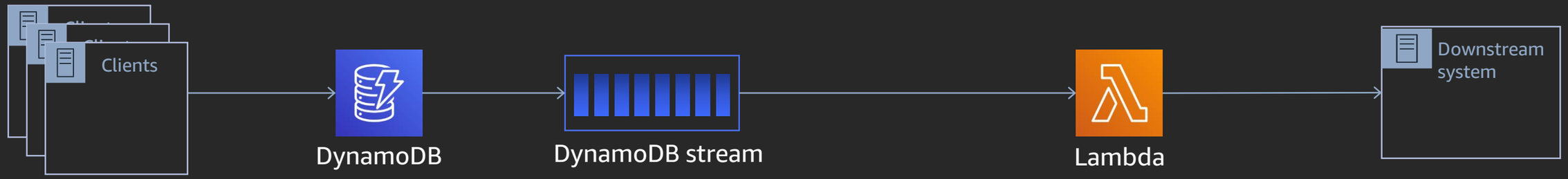
Kinesis Metrics/Alarms

- GetRecords.IteratorAgeMilliseconds
- IncomingRecords/IncomingBytes
- ReadProvisionedThroughputExceeded
- WriteProvisionedThroughputExceeded

Lambda Metrics/Alarms

- Errors
- IteratorAge
- Throttles

DynamoDB stream monitoring



DynamoDB Streams metrics/alarms

- ReturnedRecordsCount/ReturnedBytes
- UserErrors

Lambda metrics/alarms

- Errors
- IteratorAge
- Throttles
- Duration

Error handling options

NEW

A number of new options are available to tune error handling

- Maximum retry attempts – min 0, default/max 10,000
- Maximum Record Age in seconds – min 60, default/max 604,800
- Bisect Batch on Function Failure
- On-Failure Destination

Bisect Batch on Function Failure

Recursively split the failed batch and retry on a smaller subset of records, eventually isolating the problematic records

- Boolean – false by default
- These retries do NOT count towards MaximumRetryAttempts
- Make sure your function is idempotent

On-Failure Destination

An SNS Topic or SQS Queue, which is sent the metadata about a failed batch of records

- Used only after configured retry limit or maximum record age are reached.
- Remember the bisected batch retries are not counted towards retry limit.
- Does not contain the actual records, but does contain all the information needed to retrieve them!!

Common issues

Common issues

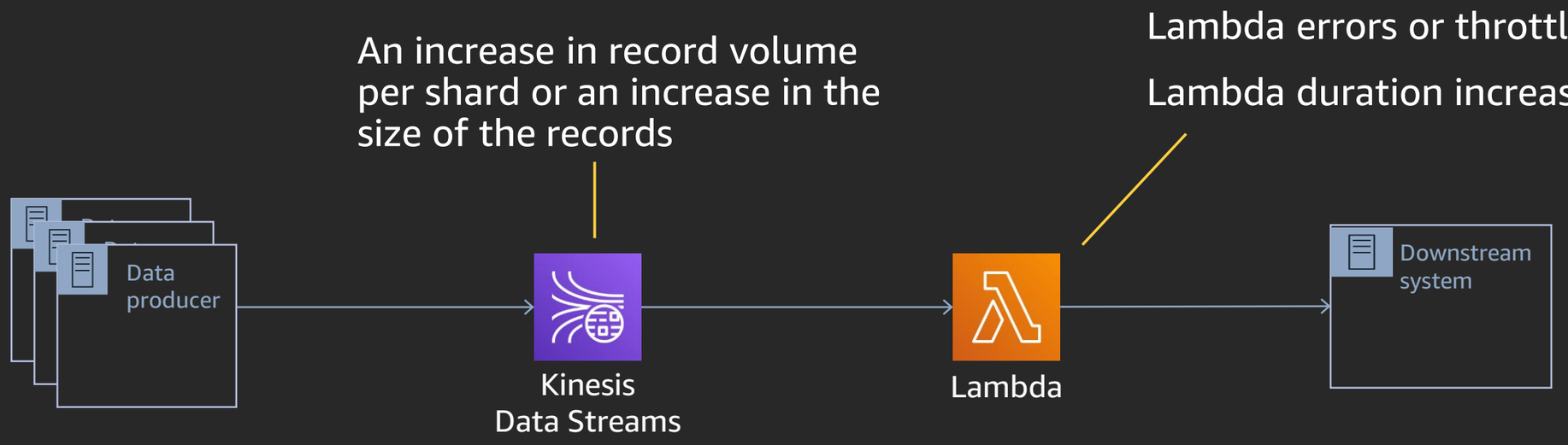
- Kinesis Data Streams
  - IteratorAge is growing rapidly
  - ReadProvisionedThroughputExceeded throttles
- DynamoDB Streams
  - IteratorAge is growing rapidly
  - Rapid growth in Lambda concurrency

Kinesis Data Streams IteratorAge is growing rapidly

- Initial questions
  - How many Lambda functions subscribed to the stream?
  - Does the Lambda function show any errors?
  - Does the Lambda function show any throttles?
  - Is there a large increase in Kinesis Data Streams metrics IncomingRecords or IncomingBytes?

Kinesis Data Streams IteratorAge is growing rapidly

Potential causes



Kinesis Data Streams IteratorAge is growing rapidly

- Solutions
  - If the Lambda is erroring
    - Configure an SQS Queue or SNS Topic for failed batches
    - Configure MaximumRetryAttempts, BisectBatchOnFunctionError, and MaximumRecordAgeInSeconds
    - Update the Lambda function to log records causing errors and return successfully
  - If the Lambda is throttling?
    - Increase per function limit/reservation, or raise the account level limit

Kinesis Data Streams IteratorAge is growing rapidly

- Solutions
  - If there is a large increase in KDS Metrics IncommingRecords or IncommingBytes
    - If this is temporary, you may be able to wait it out. Watch IteratorAge to make sure it doesn’t climb too high
    - Increase the stream data retention, this can be increased up to 7 days
    - Increase the Parallelization Factor
    - Increase the number of shards in the stream
    - Increase the memory assigned to the Lambda function or otherwise optimize the function’s performance

Kinesis Data Streams ReadProvisionedThroughputExceeded

The 5 read/sec. or 2 MiB/sec. limit is being hit

- Use enhanced fanout or remove one or more subscribers
  - Remember that Kinesis Data Firehose and Kinesis Data Analytics are subscribers as well!

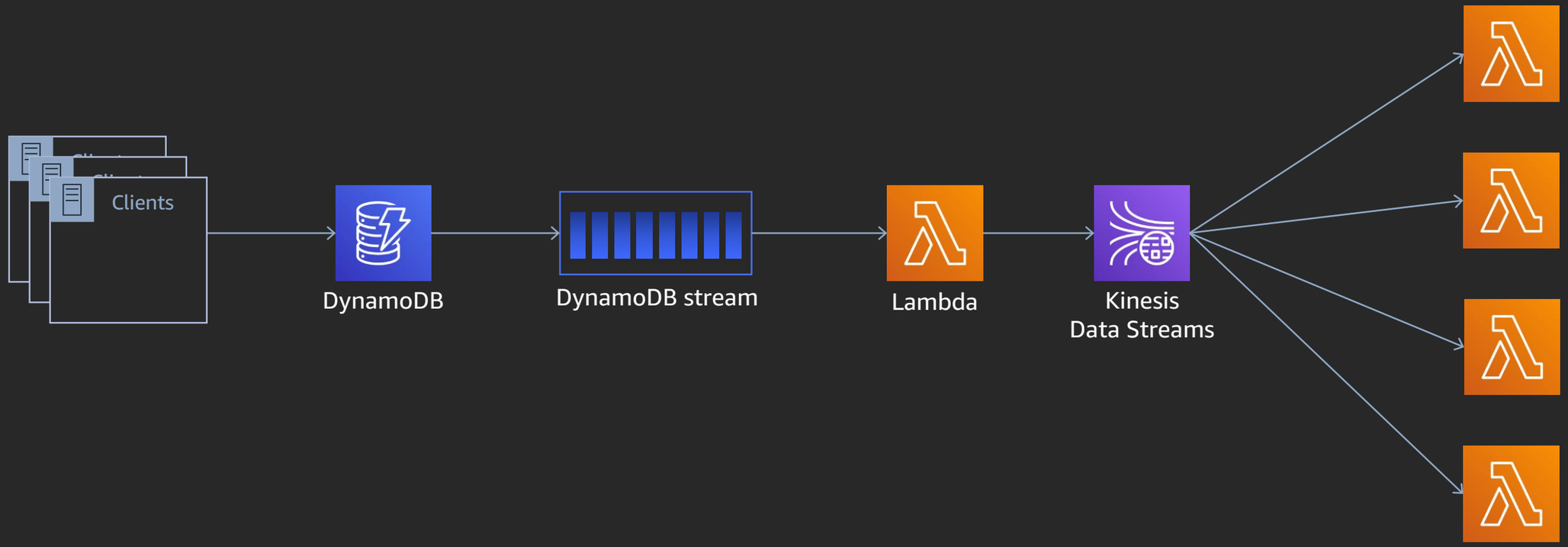
DynamoDB Streams IteratorAge is growing rapidly

- Initial questions
  - How many Lambda functions subscribed to the stream?
  - Does the Lambda function show any errors or throttles?
  - Does the Lambda function show an increase in duration?
  - Is there a large increase in the DynamoDB table write (WCU) metrics?
  - Is there a large increase in the DynamoDB stream metrics ReturnedRecordsCount or ReturnedBytes?

DynamoDB Stream IteratorAge is growing rapidly

- Solutions
  - If there is a large increase in writes on the DDB Table:
    - If this is temporary, you may be able to wait it out. Watch IteratorAge to make sure it doesn’t climb too high
    - Unlike KDS you can NOT increase the data retention time, so you need to take action more quickly
    - Increase the memory assigned to the Lambda function or otherwise optimize the function’s performance
    - Increase the Parallelization Factor
  - If there are more than two Lambda functions subscribed to the stream, consider adding a Kinesis Data Stream for increasing the fan-out

DynamoDB stream fanout



Performance and optimization

Performance

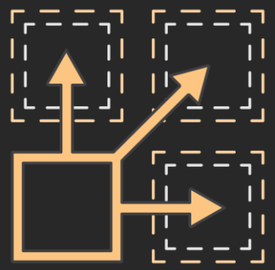
- What matters to your application?
  - End-to-end latency
  - Overall cost
- Kinesis Data Streams enhanced fanout (EFO)
- DynamoDB Streams
- Small messages in Kinesis Data Streams
  - Aggregation/de-aggregation libraries
  - Compression
- Low throughput streams—batch window to the rescue

Lambda supports Kinesis Data Streams Enhanced Fan-Out and HTTP/2 for faster streaming

Enhanced fan-out allows customers to scale the number of functions reading from a stream in parallel while maintaining performance

HTTP/2 data retrieval API improves data delivery speed between data producers and Lambda functions by more than 65%

Kinesis Data Streams: Enhanced Fan-Out



When to use standard consumers:

- Total number of consuming applications is low (< 3)
- Consumers are not latency-sensitive
- Newer error handling options are needed*
- Minimize cost

When to use Enhanced Fan-Out consumers:

- Multiple consumer applications for the same Kinesis Data Stream
    - Default limit of 5 registered consuming applications. More can be supported with a service limit increase request
- Low-latency requirements for data processing
    - Messages are typically delivered to a consumer in less than 70 ms

*Note that today EFO does not have the advanced Error handling options

Optimizing Small Messages in Kinesis

- Kinesis Data Streams per shard write limits
  - 1MiB/sec or 1,000 messages/sec
  - With high volumes of small messages you reach the 1,000 messages/sec limit easily
  - This leads to lower throughput per shard and higher costs

Aggregation is the answer!

Aggregation / de-aggregation options

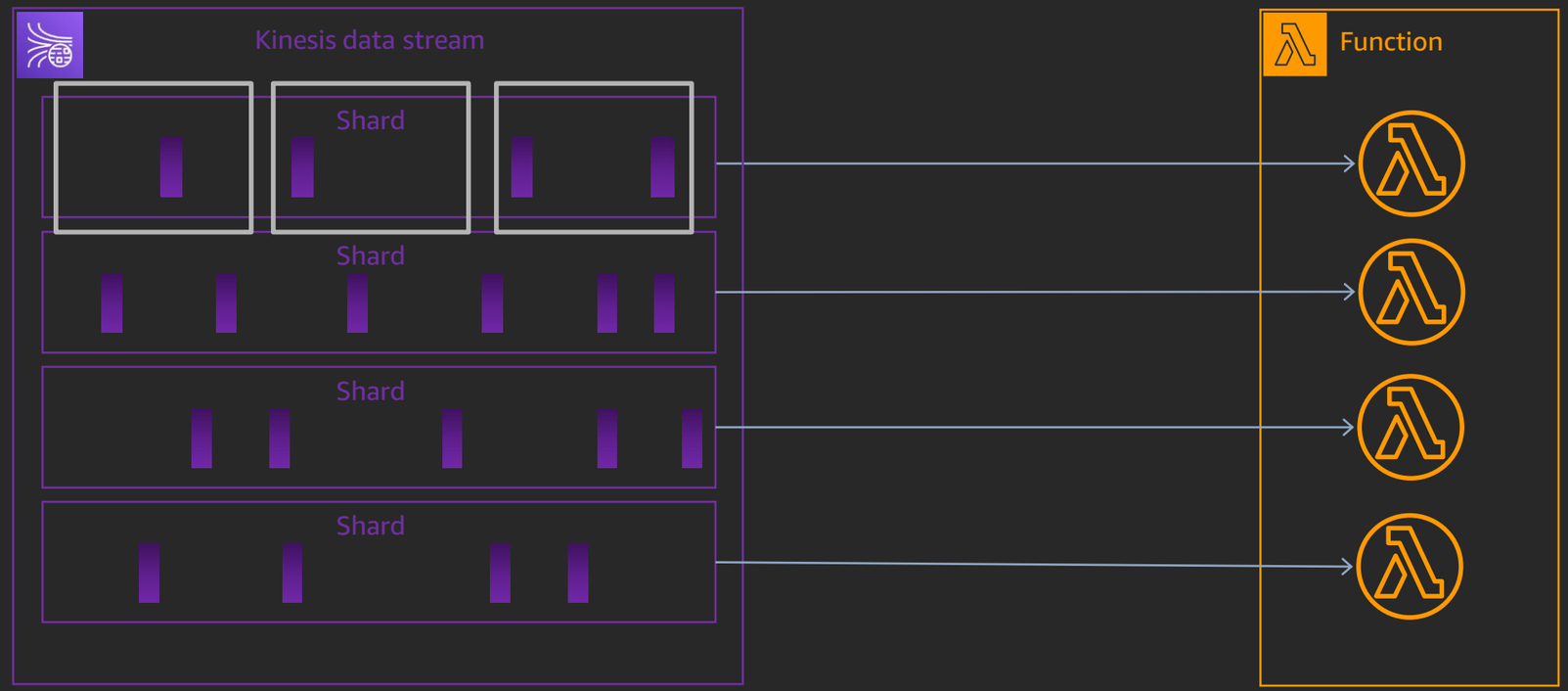
- Producer side
  - Kinesis Producer Library(KPL)(https://github.com/awslabs/amazon-kinesis-producer)
  - Kinesis Aggregation Library(https://github.com/awslabs/kinesis-aggregation)
- Consumer side within Lambda
  - Kinesis Aggregation Library(https://github.com/awslabs/kinesis-aggregation)
    - Java, Node.js, and Python versions available
- Another option if your data has a consistent format is Avro

Low-throughput streams

Lambda triggered with very small batches

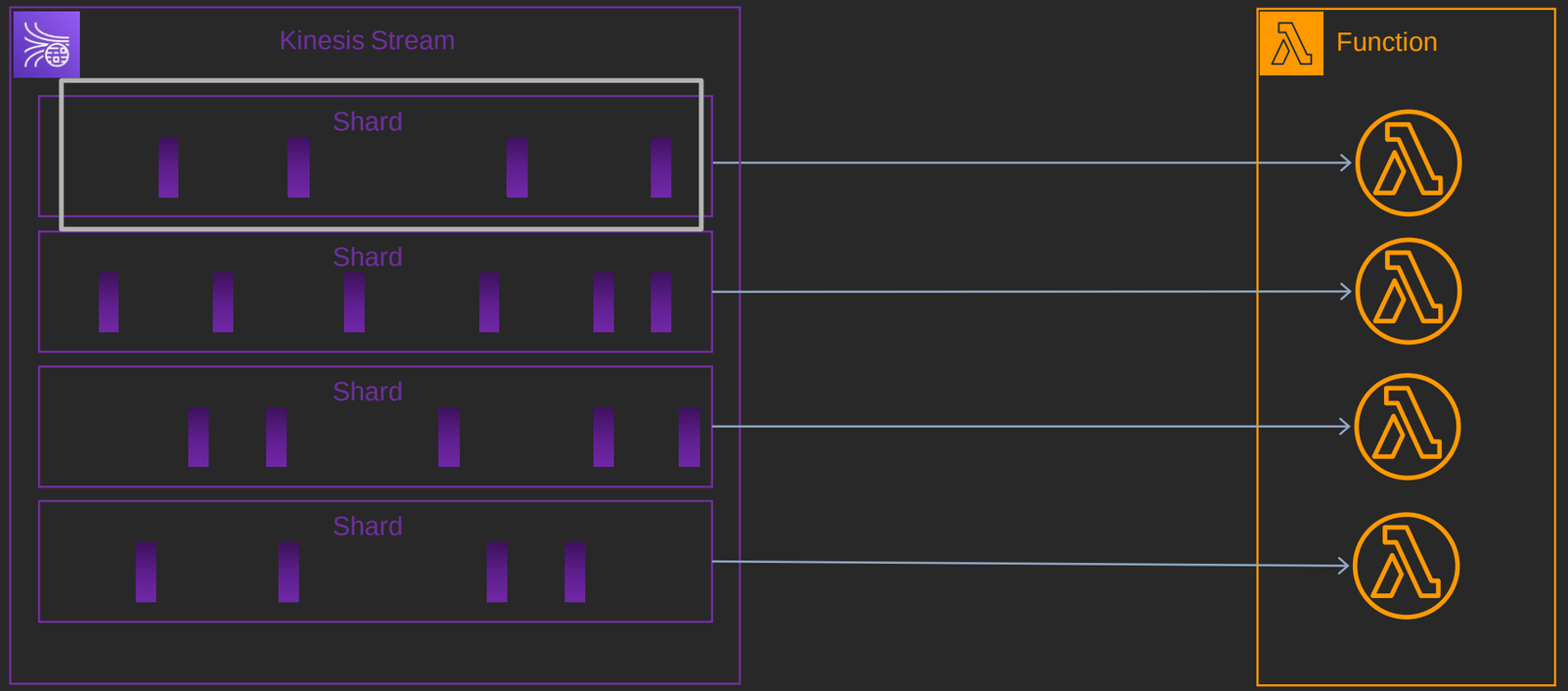
Leads to higher cost per message

For archiving workloads the resulting payload is too small



Batch window

- Additional knob to tune the stream trigger
  - Set a time to wait before triggering. Max five minutes, set in seconds.
  - Batch size is still respected and will trigger on full batches before the batch window is up
  - Works for both Kinesis Data Streams and DynamoDB Streams triggers

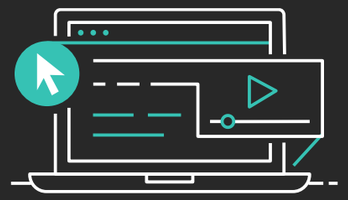


Conclusion

- Be clear on the goals of your streaming system
- Understand how your system scales
- Prepare for failures, make use of the new rror handling options
- Test individual components as well as end to end

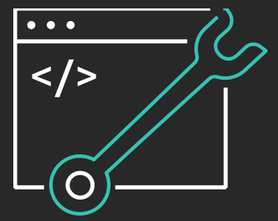
Learn serverless with AWS Training and Certification

Resources created by the experts at AWS to help you learn modern application development



Free, on-demand courses on serverless, including

- Introduction to Serverless Development
- Getting into the Serverless Mindset
- AWS Lambda Foundations
- Amazon API Gateway for Serverless Applications
- Amazon DynamoDB for Serverless Architectures



Additional digital and classroom trainings cover modern application development and computing

Visit the Learning Library at https://aws.training

aws training and certification

Thank you!

AWS re:Invent

aws

Please complete the session survey in the mobile app.