
AWS
re:Invent

SVS407

Deep dive on AWS Lambda consumers for Amazon Kinesis

Heeki Park

Principal Solutions Architect
Serverless Specialist
AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Deep dive on AWS Lambda consumers for Amazon Kinesis

Producers and records

Amazon Kinesis Data Streams

Standard Lambda consumers

Parallelization factor

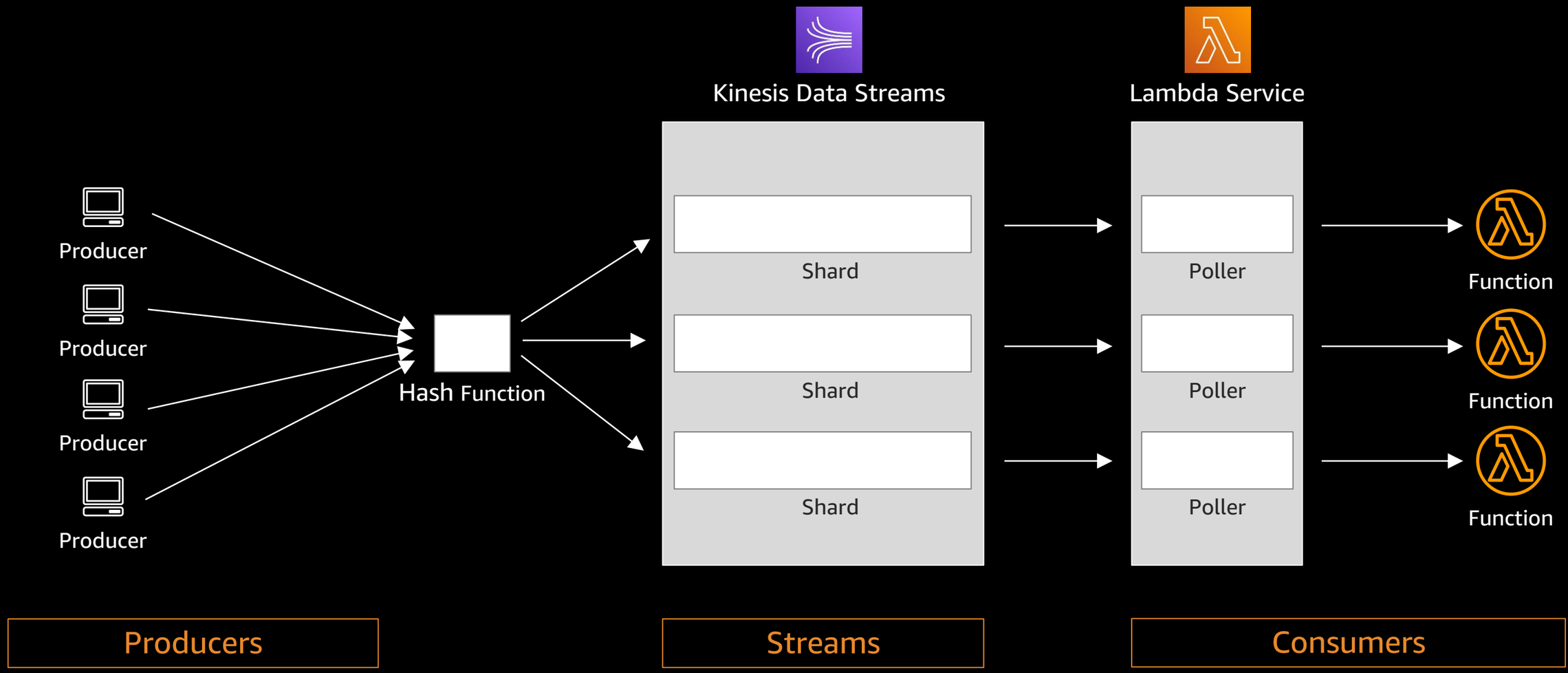
Enhanced fan-out (EFO) consumers



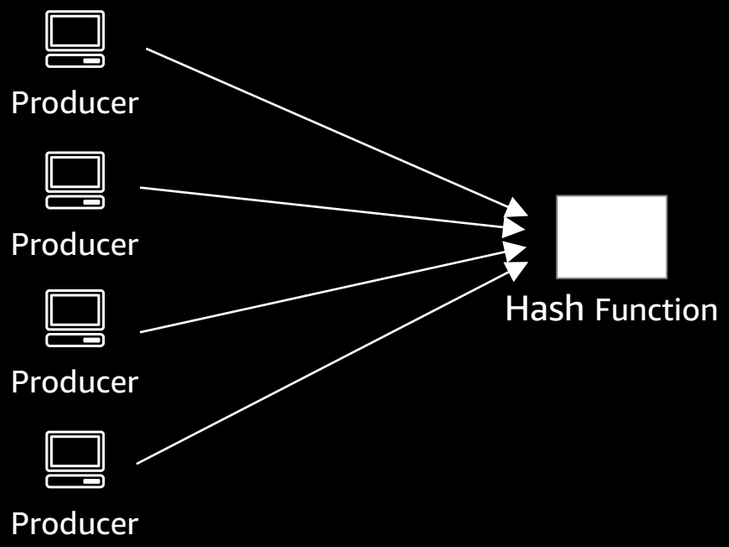
AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Overview of message flow



Producers



Producers

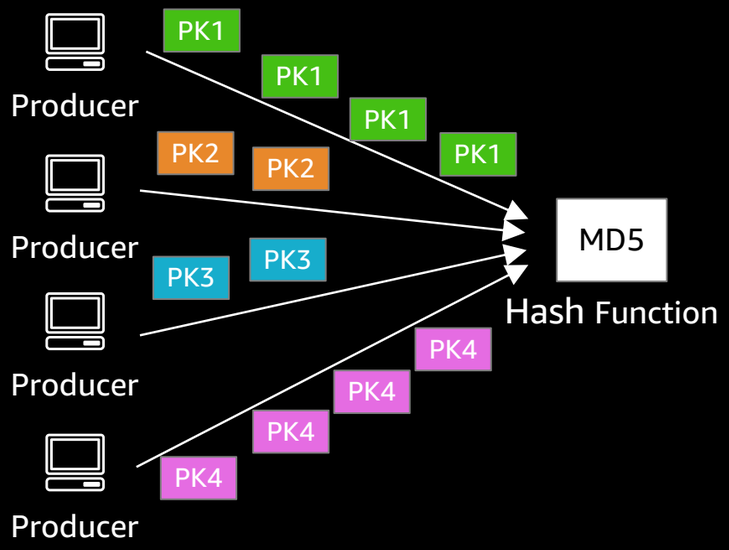
Could be a web server sending logs
Could be application servers sending metrics
Could be Internet of Things (IoT) devices sending telemetry

Put records into streams with:

```
client.putRecord()
client.putRecords()
```

Or use the Kinesis Producer Library

Records

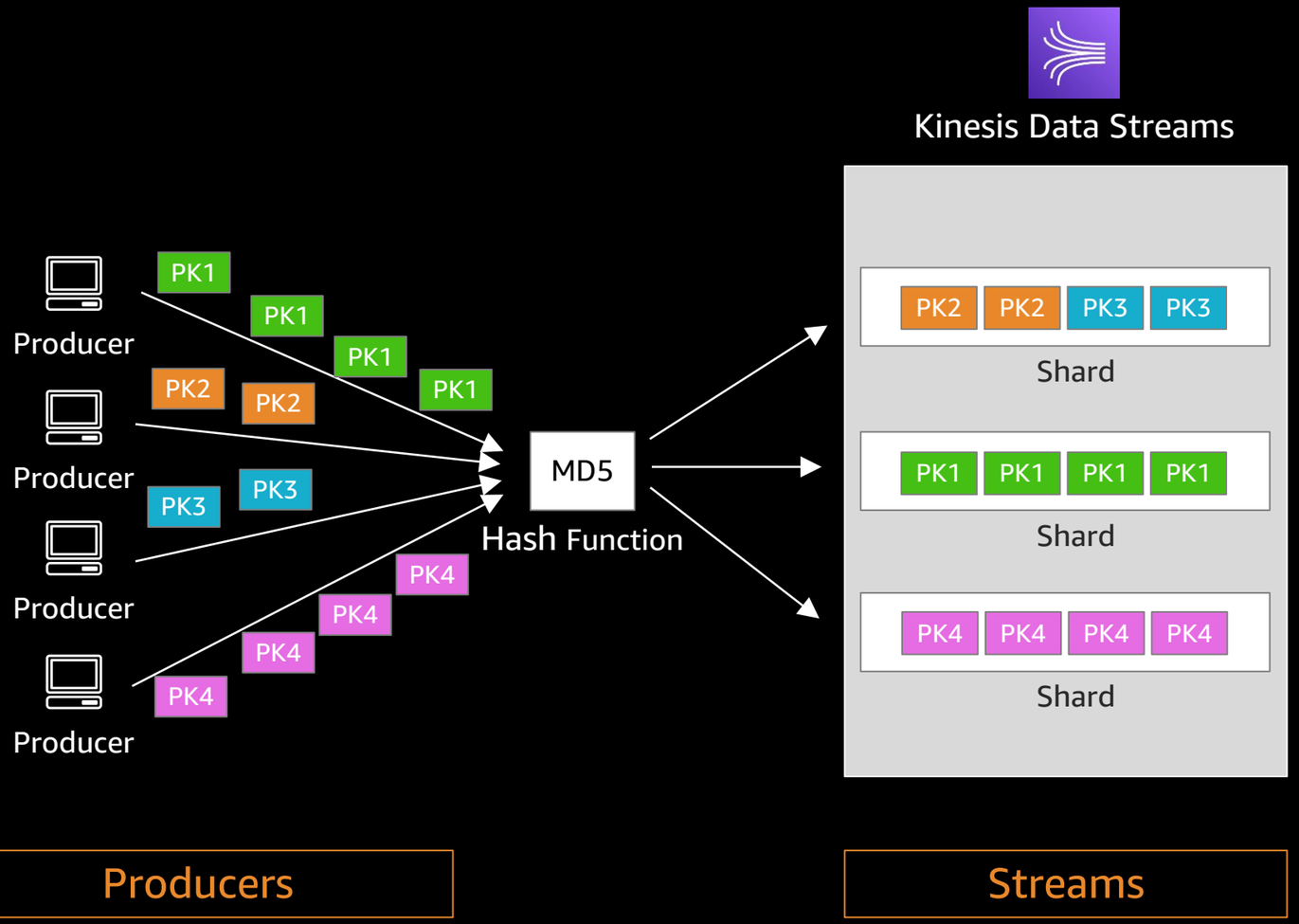


Producers

```
{
  “partitionKey”: “key1”,
  “sequenceNumber”: “12345678”,
  “data”: “base64_encoded_payload”
  ...
}
```

Partition key is hashed to route to a shard
Sequence number for strict ordering
Messages up to 1 MB in size

Streams



Stream is made of shards
Order maintained within shards

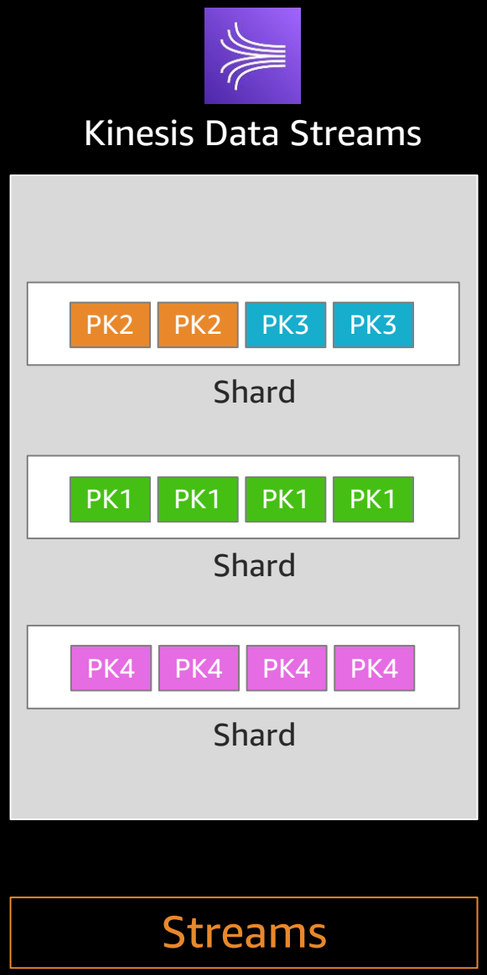
```
total_stream_cap = 0
for shard in stream:
  total_stream_cap += shard.cap
```

Retain records up to 1 year

Shards

Reads support up to
5 requests or
2 MB per second

Writes support up to
1,000 records or
1 MB per second



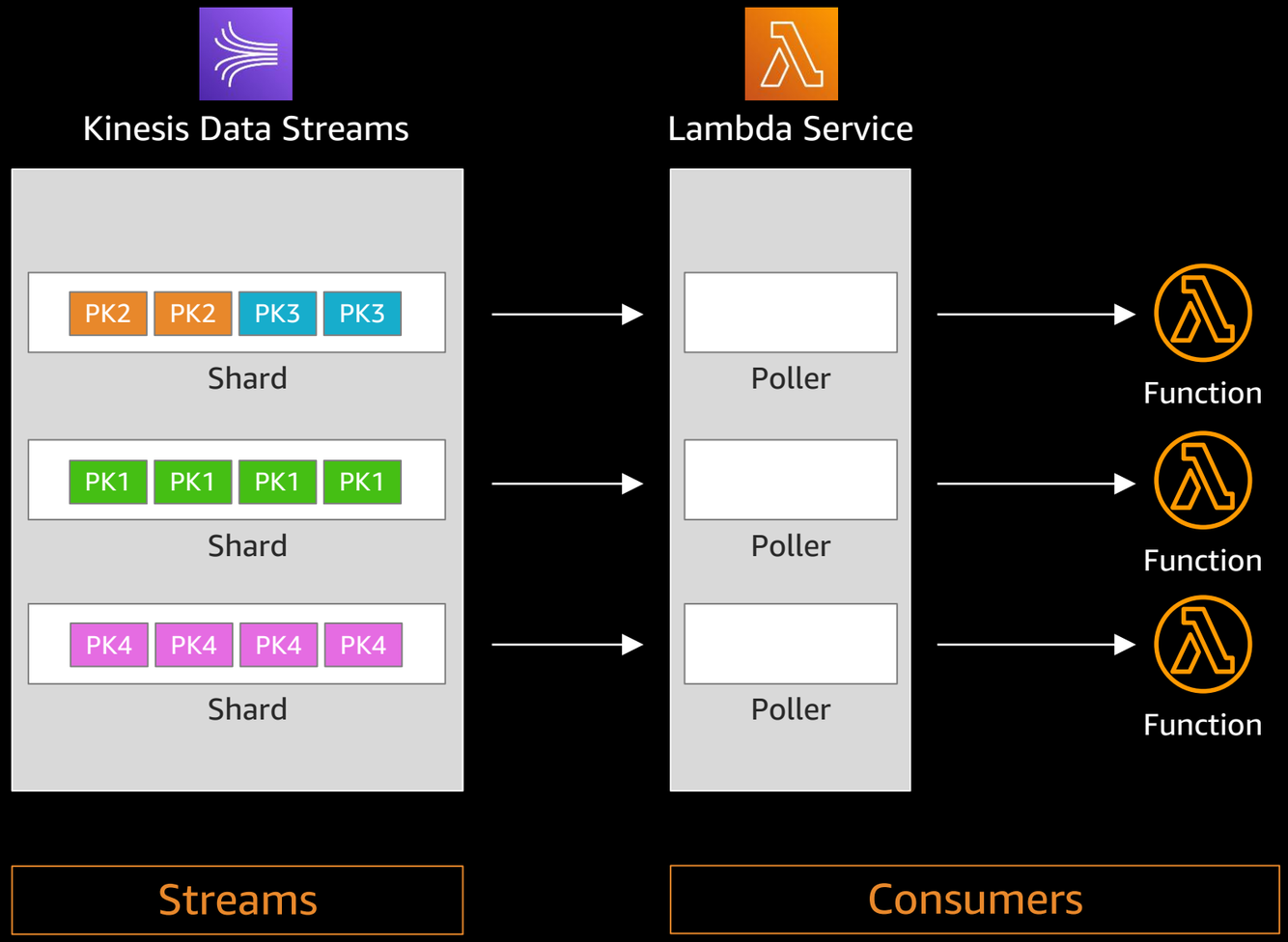
In this stream with 3 shards,
assuming equal distribution

Aggregate reads yield
15 requests or
6 MB per second

Aggregate writes yield
3,000 records or
3 MB per second

Pollers as consumers

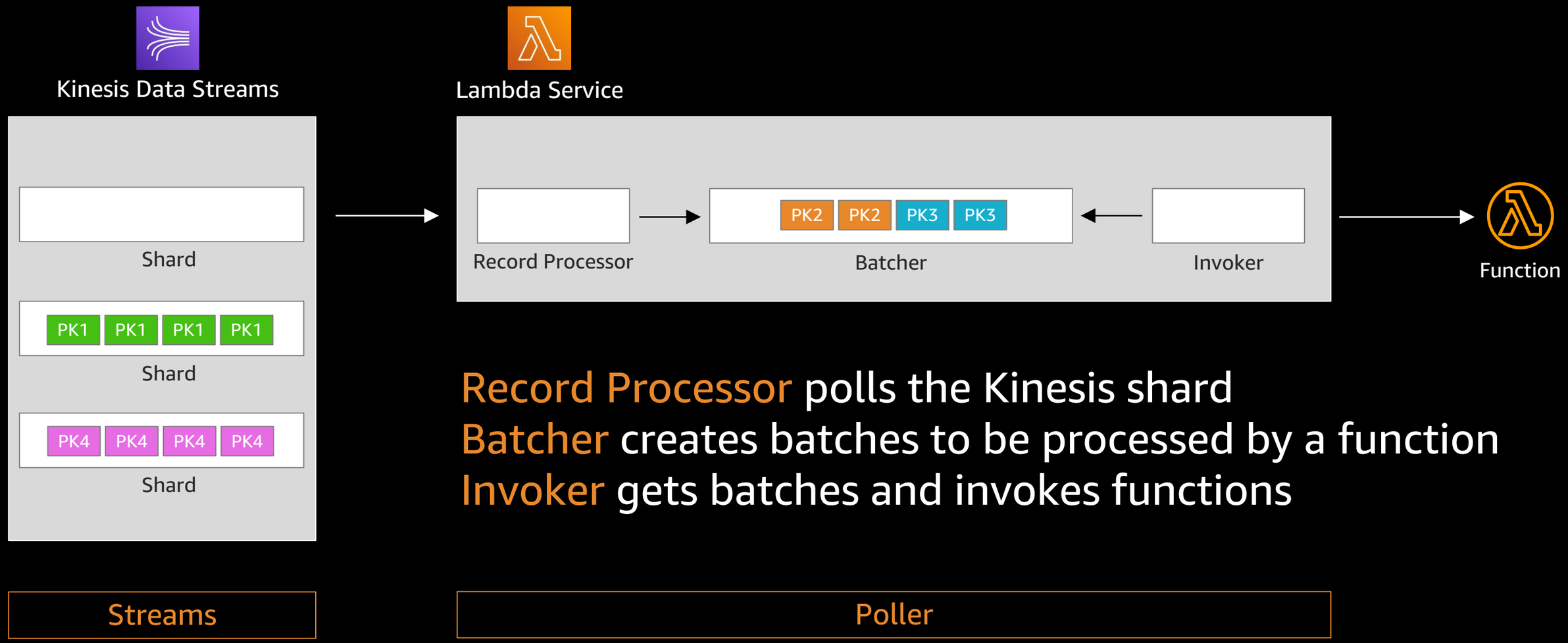
Lambda configures a poller automatically that serves as the intermediate consumer, which polls every 1 second



Lambda event payload


```
{
  "Records": [
    {
      "kinesis": {
        "kinesisSchemaVersion": "1.0",
        "partitionKey": "1",
        "sequenceNumber": "49590338271490256608559692538361571095921575989136588898",
        "data": "SGVsbG8sIHRoaXMgaXMgYSB0ZXN0Lg==",
        "approximateArrivalTimestamp": 1545084650.987
      },
      "eventSource": "aws:kinesis",
      "eventVersion": "1.0",
      "eventID": "shardId-000000000006:49590338271490256608559692538361571095921575989136588898",
      "eventName": "aws:kinesis:record",
      "invokeIdentityArn": "arn:aws:iam::123456789012:role/lambda-role",
      "awsRegion": "us-east-2",
      "eventSourceARN": "arn:aws:kinesis:us-east-2:123456789012:stream/lambda-stream"
    },
    ...
  ]
}
```

Poller in detail



Record Processor polls the Kinesis shard
Batcher creates batches to be processed by a function
Invoker gets batches and invokes functions

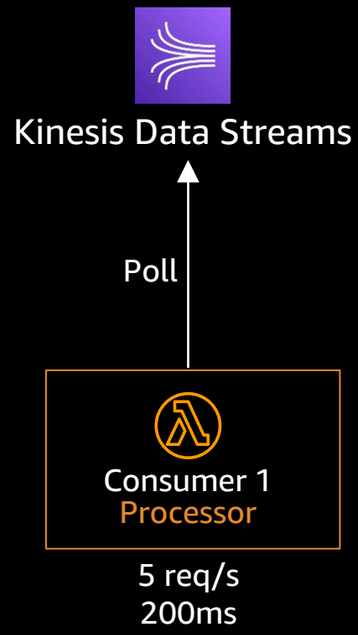
An aside on consumers

Standard consumers

Read throughput shared across all consumers

Average 200 ms latency for single consumer; 1,000 ms if five consumers

Pull model over HTTP



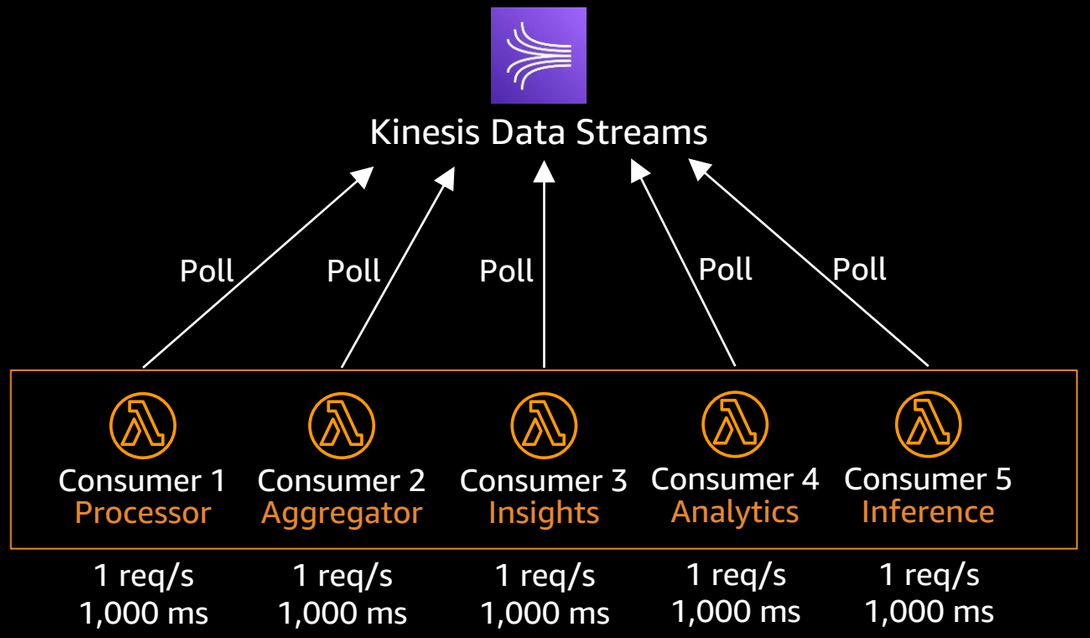
An aside on consumers

Standard consumers

Read throughput shared across all consumers

Average 200 ms latency for single consumer; 1,000 ms if five consumers

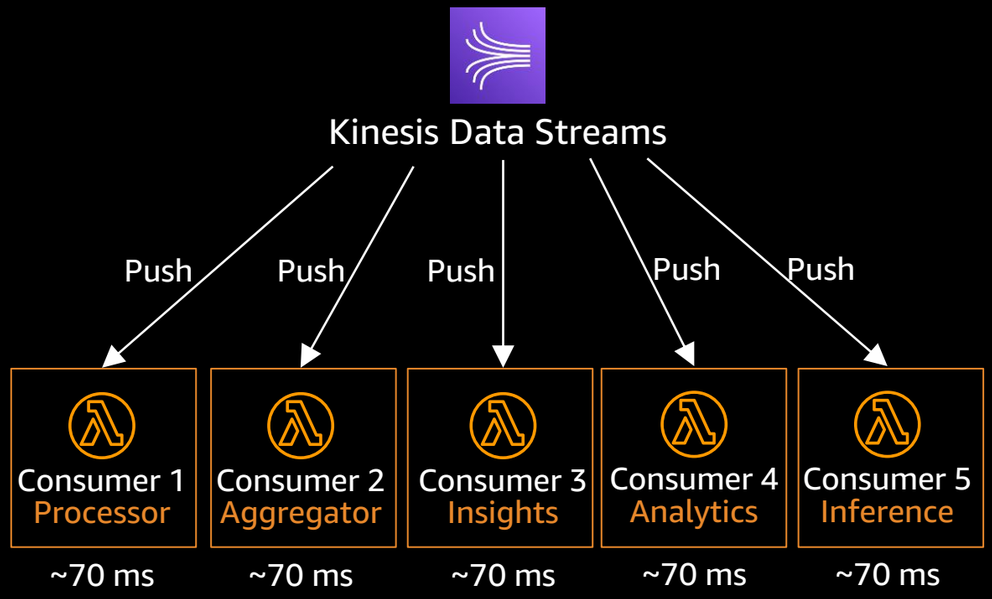
Pull model over HTTP



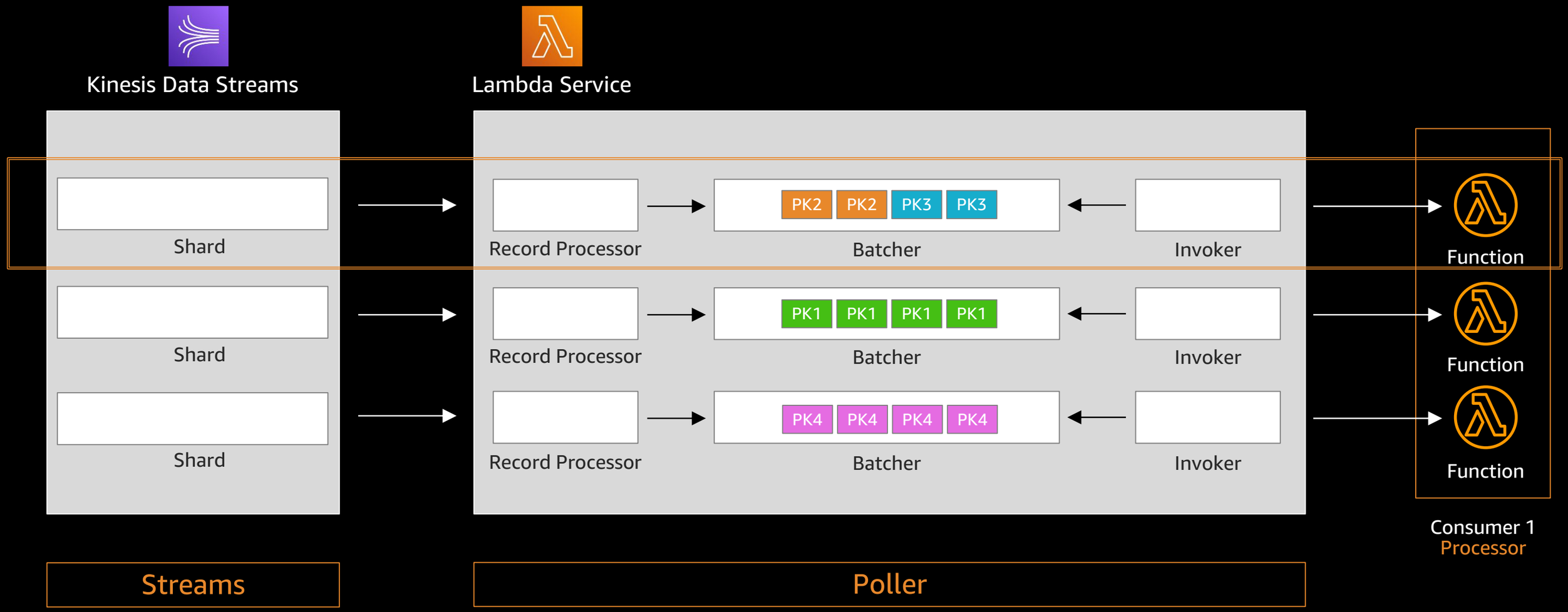
An aside on consumers

EFO consumers

- Read throughput dedicated for each consumer
- Average 70 ms latency for each consumer, independent of other consumers
- Push model over HTTP/2
- Charged hourly charge and per GB retrieved



Standard consumers with Lambda



Standard consumers with Lambda

```
FnMapping:
  Type: AWS::Lambda::EventSourceMapping
  Properties:
    BatchSize: 100
    BisectBatchOnFunctionError: true
    DestinationConfig:
      OnFailure:
        Destination: !GetAtt DLQ.Arn
    Enabled: true
    EventSourceArn: !Ref kinesisArn
    FunctionName: !Ref Fn
    MaximumBatchingWindowInSeconds: 60
    StartingPosition: LATEST
```

Specifies the Amazon Resource Name (ARN) of the event source, a Kinesis stream, in this example

Standard consumers with Lambda

```
FnMapping:
  Type: AWS::Lambda::EventSourceMapping
  Properties:
    BatchSize: 100
    BisectBatchOnFunctionError: true
    DestinationConfig:
      OnFailure:
        Destination: !GetAtt DLQ.Arn
    Enabled: true
    EventSourceArn: !Ref kinesisArn
    FunctionName: !Ref Fn
    MaximumBatchingWindowInSeconds: 60
    StartingPosition: LATEST
```

Specifies the function that will be invoked

Standard consumers with Lambda

```
FnMapping:
  Type: AWS::Lambda::EventSourceMapping
  Properties:
    BatchSize: 100
    BisectBatchOnFunctionError: true
    DestinationConfig:
      OnFailure:
        Destination: !GetAtt DLQ.Arn
    Enabled: true
    EventSourceArn: !Ref kinesisArn
    FunctionName: !Ref Fn
    MaximumBatchingWindowInSeconds: 60
    StartingPosition: LATEST
```

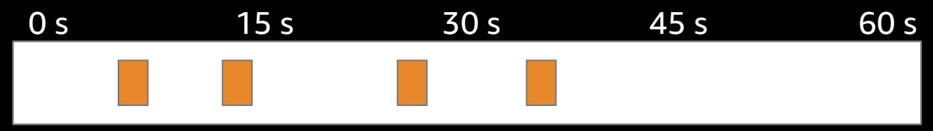
Specifies the maximum amount of time to gather records before invoking a function

Standard consumers with Lambda

```
FnMapping:
  Type: AWS::Lambda::EventSourceMapping
  Properties:
    BatchSize: 100
    BisectBatchOnFunctionError: true
    DestinationConfig:
      OnFailure:
        Destination: !GetAtt DLQ.Arn
    Enabled: true
    EventSourceArn: !Ref kinesisArn
    FunctionName: !Ref Fn
    MaximumBatchingWindowInSeconds: 60
    StartingPosition: LATEST
```

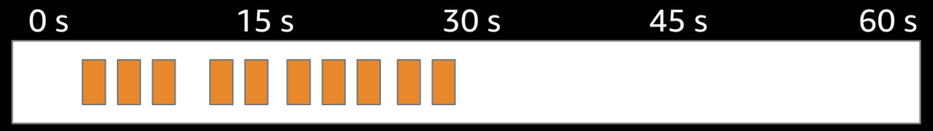
Specifies the number of records that the poller's batcher will gather before an invoke is triggered

BatchSize vs. MaximumBatchWindowInSeconds



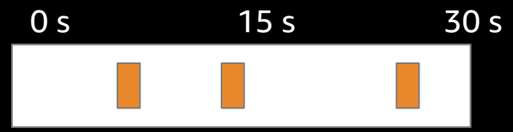
BatchSize = 10, MaximumBatchWindowInSeconds = 60

→ Invokes after 60 s



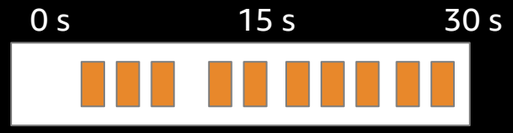
BatchSize = 10, MaximumBatchWindowInSeconds = 60

→ Invokes with 10 records (~30 s)



BatchSize = 10, MaximumBatchWindowInSeconds = 30

→ Invokes after 30 s



BatchSize = 5, MaximumBatchWindowInSeconds = 30

→ Invokes with 5 records (~15 s)
Invokes with 5 records (~30 s)

What if the applications needs to process records faster?

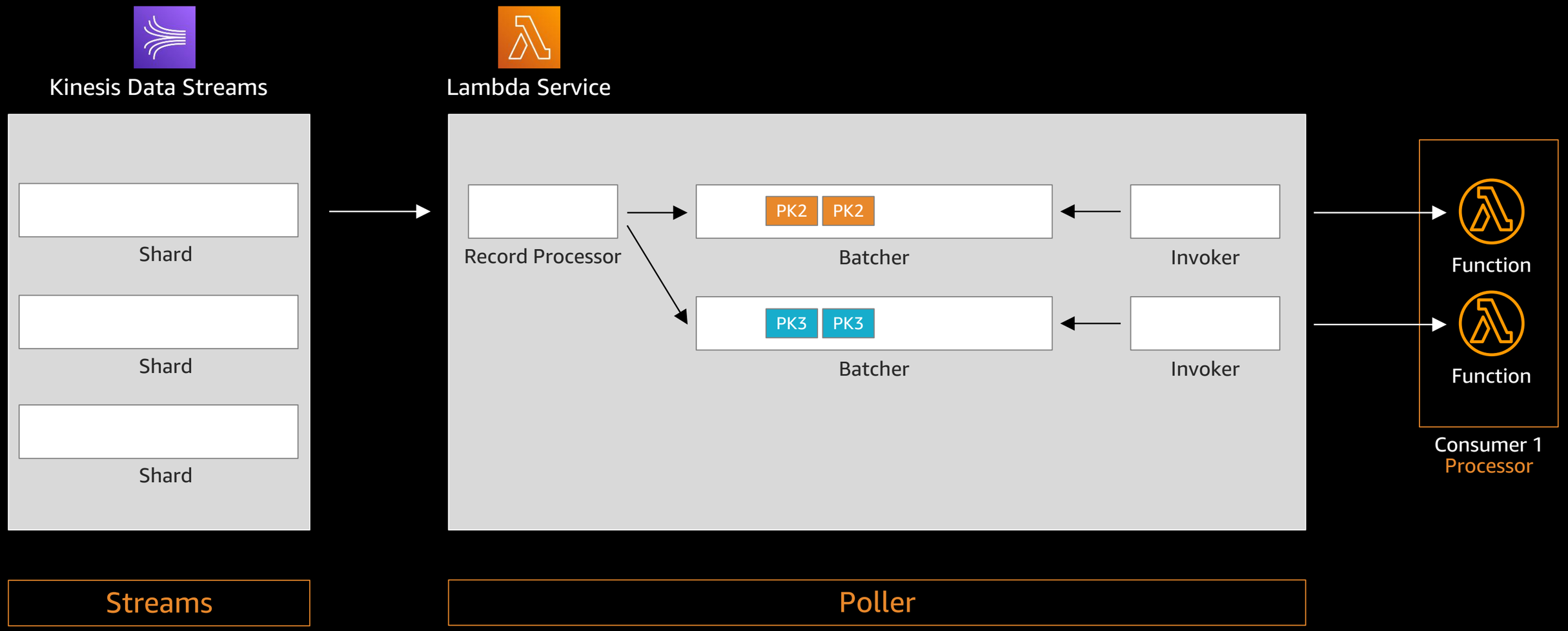
Could decrease the maximum batch window

Could reduce batch size to invoke more frequently

Could increase memory of the function to increase compute power

Could increase number of shards for more throughput

Standard consumers with Lambda parallelized



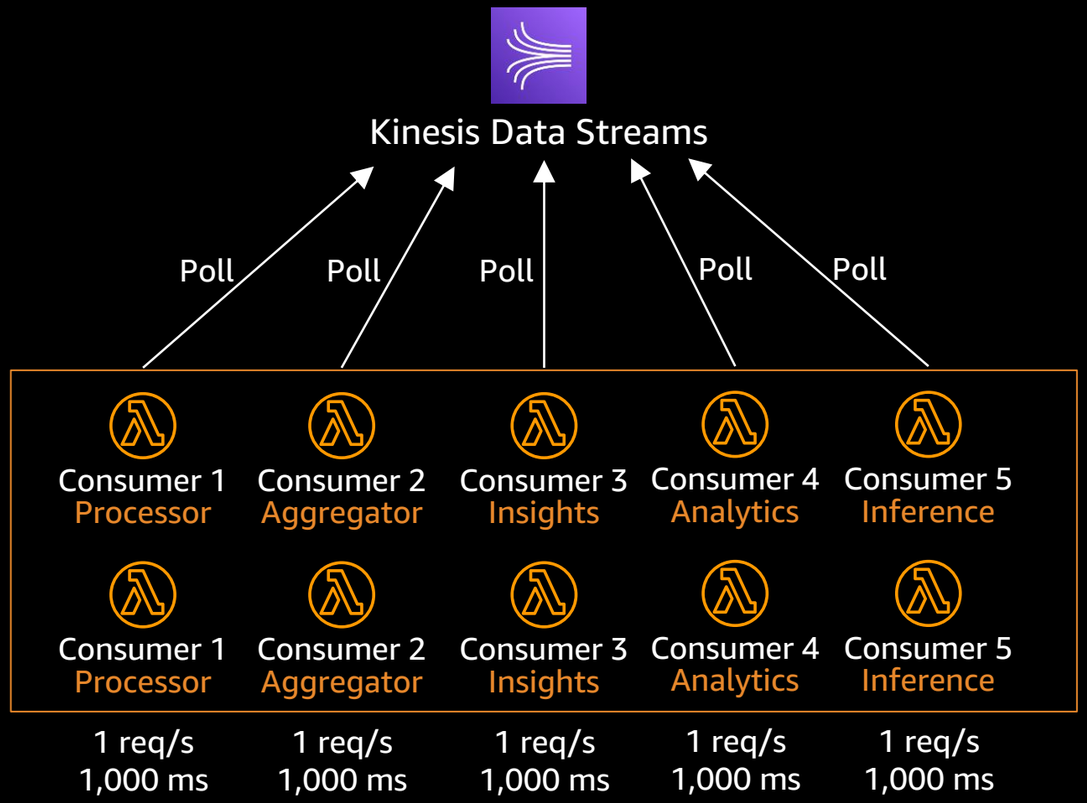
Consumers parallelized

Standard consumers

Read throughput shared across all consumers

Average 200 ms latency for single consumer; 1,000 ms if five consumers

Pull model over HTTP

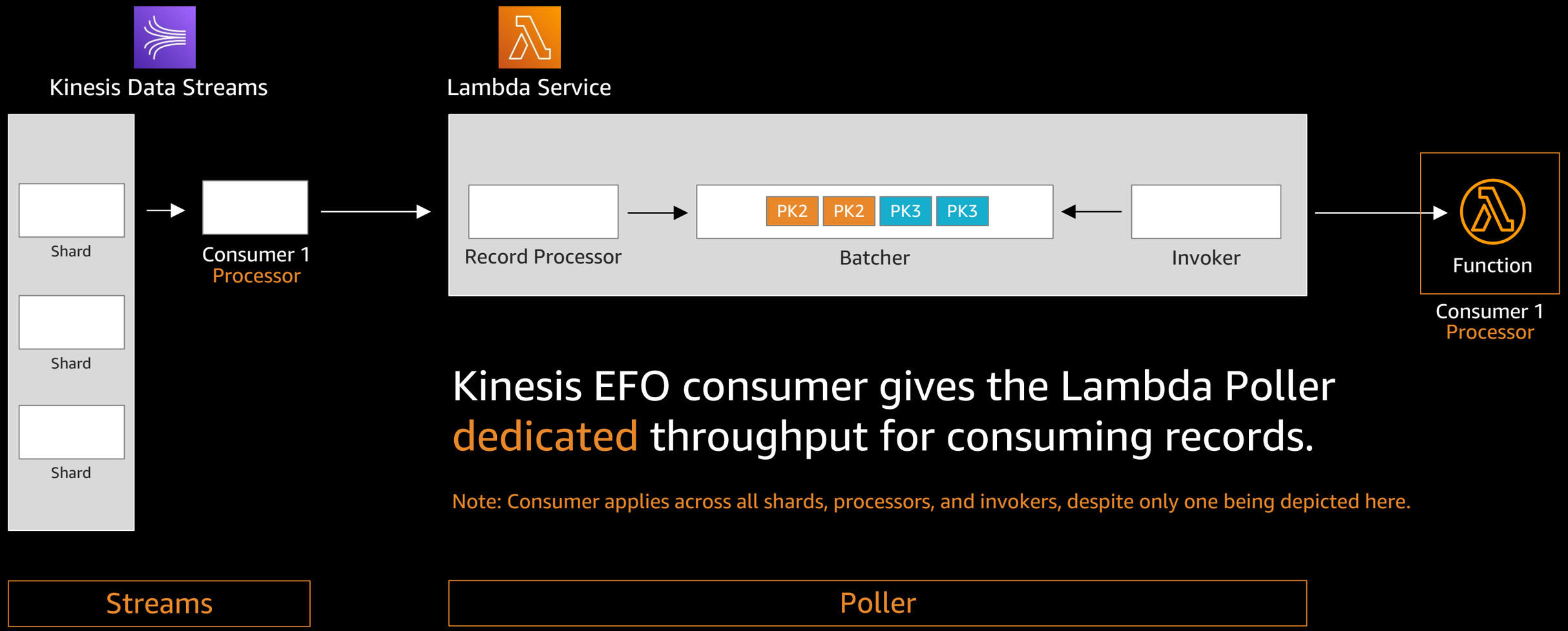


Standard consumers with Lambda parallelized

```
FnMapping:
  Type: AWS::Lambda::EventSourceMapping
  Properties:
    BatchSize: 100
    BisectBatchOnFunctionError: true
    DestinationConfig:
      OnFailure:
        Destination: !GetAtt DLQ.Arn
    Enabled: true
    EventSourceArn: !Ref kinesisArn
    FunctionName: !Ref Fn
    MaximumBatchingWindowInSeconds: 60
    ParallelizationFactor: 2
    StartingPosition: LATEST
```

Specifies the number of parallel invocations for each shard

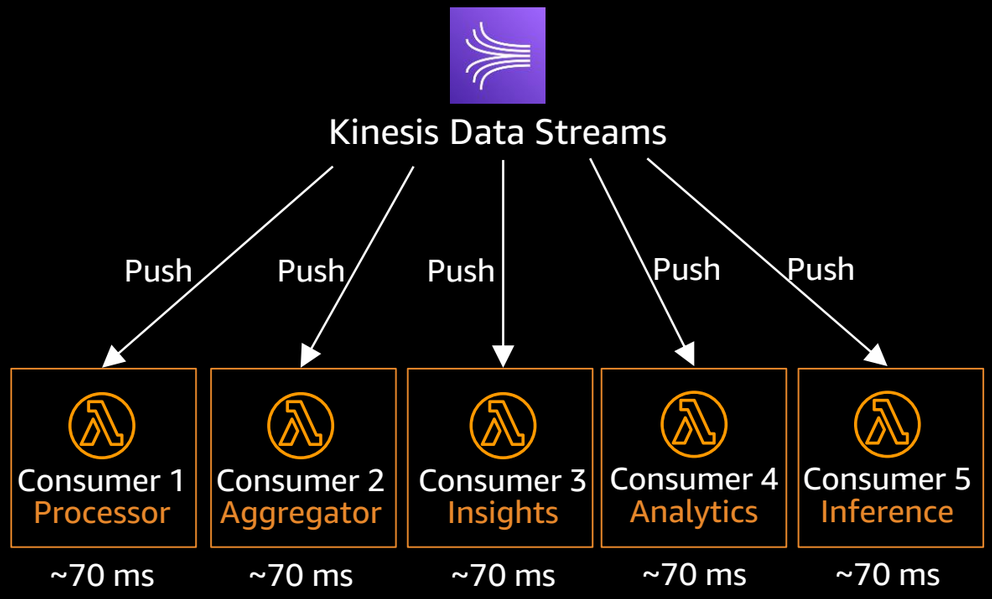
EFO consumers with Lambda



EFO consumer revisited

EFO consumers

- Read throughput dedicated for each consumer
- Average 70 ms latency for each consumer, independent of other consumers
- Push model over HTTP/2
- Charged hourly charge and per GB retrieved



EFO consumer

```
Consumer:
  Type: AWS::Kinesis::StreamConsumer
  Properties:
    ConsumerName: ‘my-consumer’
    StreamARN: !Ref kinesisArn
```

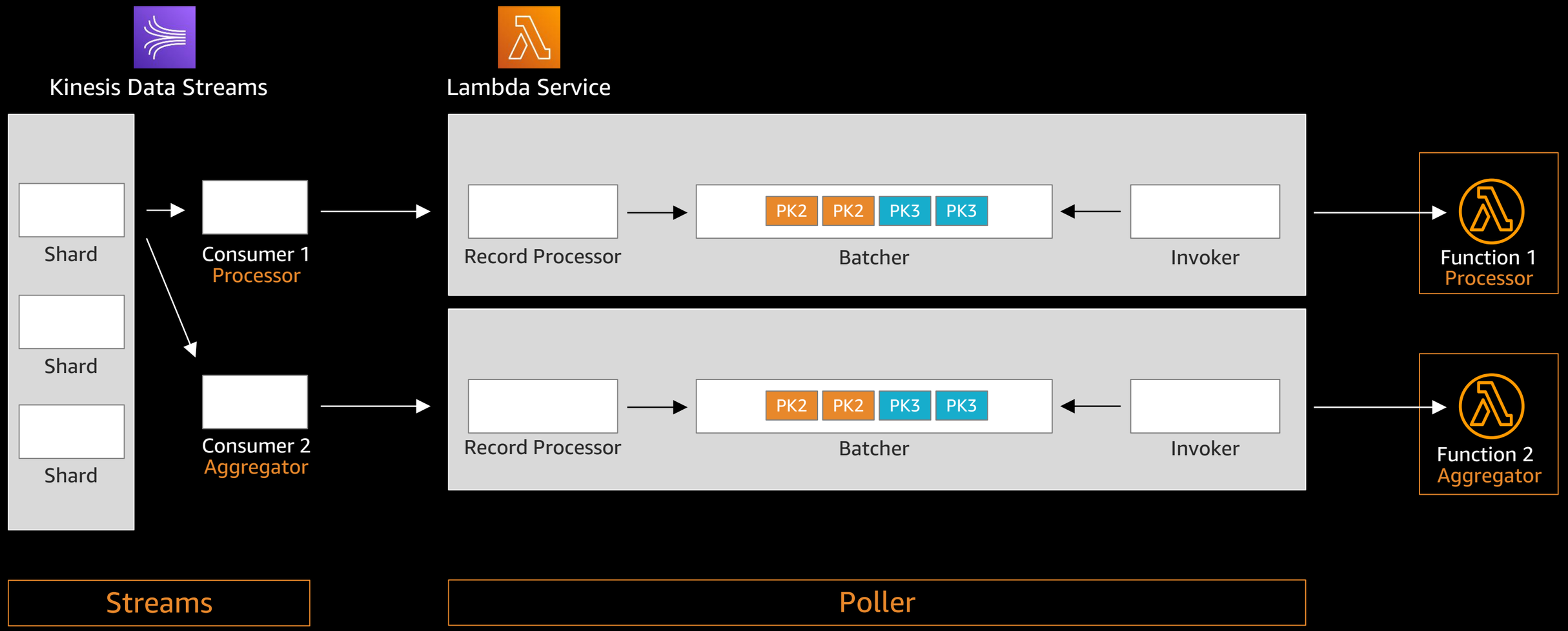
Specifies the ARN of the stream and will then attach the EventSourceMapping to this consumer

EFO consumers with Lambda

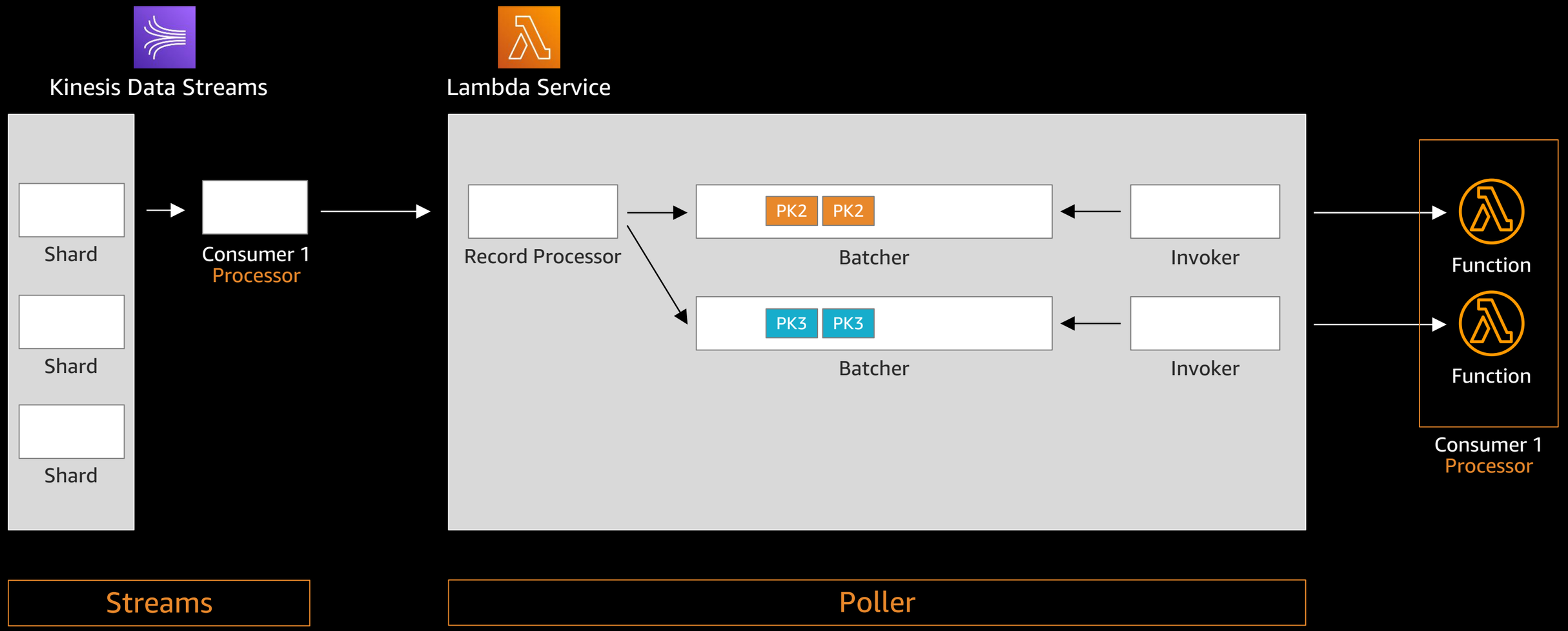
```
FnMapping:
  Type: AWS::Lambda::EventSourceMapping
  Properties:
    BatchSize: 100
    BisectBatchOnFunctionError: true
    DestinationConfig:
      OnFailure:
        Destination: !GetAtt DLQ.Arn
    Enabled: true
    EventSourceArn: !Ref Consumer
    FunctionName: !Ref Fn
    MaximumBatchingWindowInSeconds: 60
    StartingPosition: LATEST
```

Specifies the ARN of the event source, updated to the Kinesis consumer

EFO consumers with Lambda



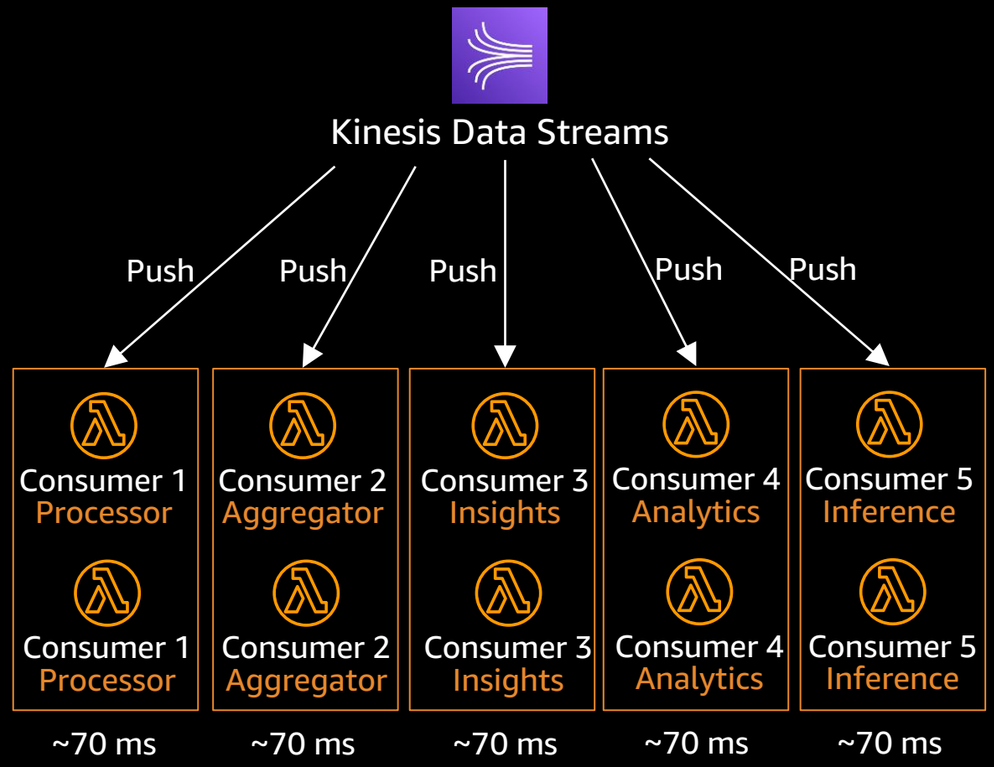
EFO consumers with Lambda Parallelized



EFO consumer parallelized

EFO consumers

- Read throughput dedicated for each consumer
- Average 70 ms latency for each consumer, independent of other consumers
- Push model over HTTP/2
- Charged hourly charge and per GB retrieved



EFO consumers with Lambda parallelized

```
FnMapping:
  Type: AWS::Lambda::EventSourceMapping
  Properties:
    BatchSize: 100
    BisectBatchOnFunctionError: true
    DestinationConfig:
      OnFailure:
        Destination: !GetAtt DLQ.Arn
    Enabled: true
    EventSourceArn: !Ref Consumer
    FunctionName: !Ref Fn
    MaximumBatchingWindowInSeconds: 60
    ParallelizationFactor: 2
    StartingPosition: LATEST
```

Specifies the ARN of the event source, updated to the Kinesis consumer

Specifies the number of parallel invokes for each shard

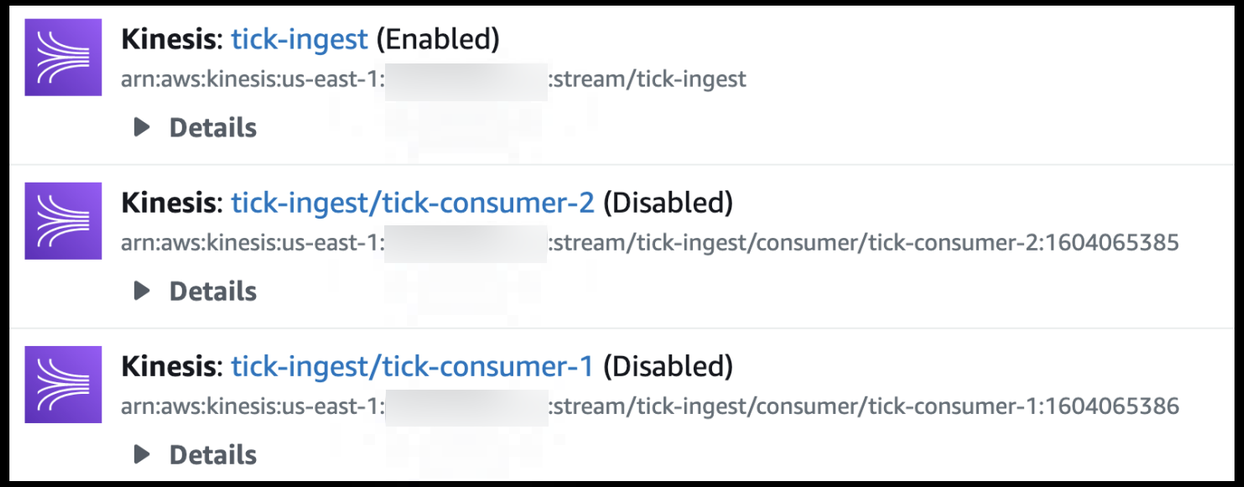
Why all these options?

May need a single application to process records quickly in parallel

May need multiple applications to process records with dedicated throughput

May even need multiple applications to process records, each with parallel processing, with dedicated throughput

Show me!



Stream configured with 4 shards

Standard consumer (enabled)

EFO consumer 1 (disabled)

EFO consumer 2 (disabled)

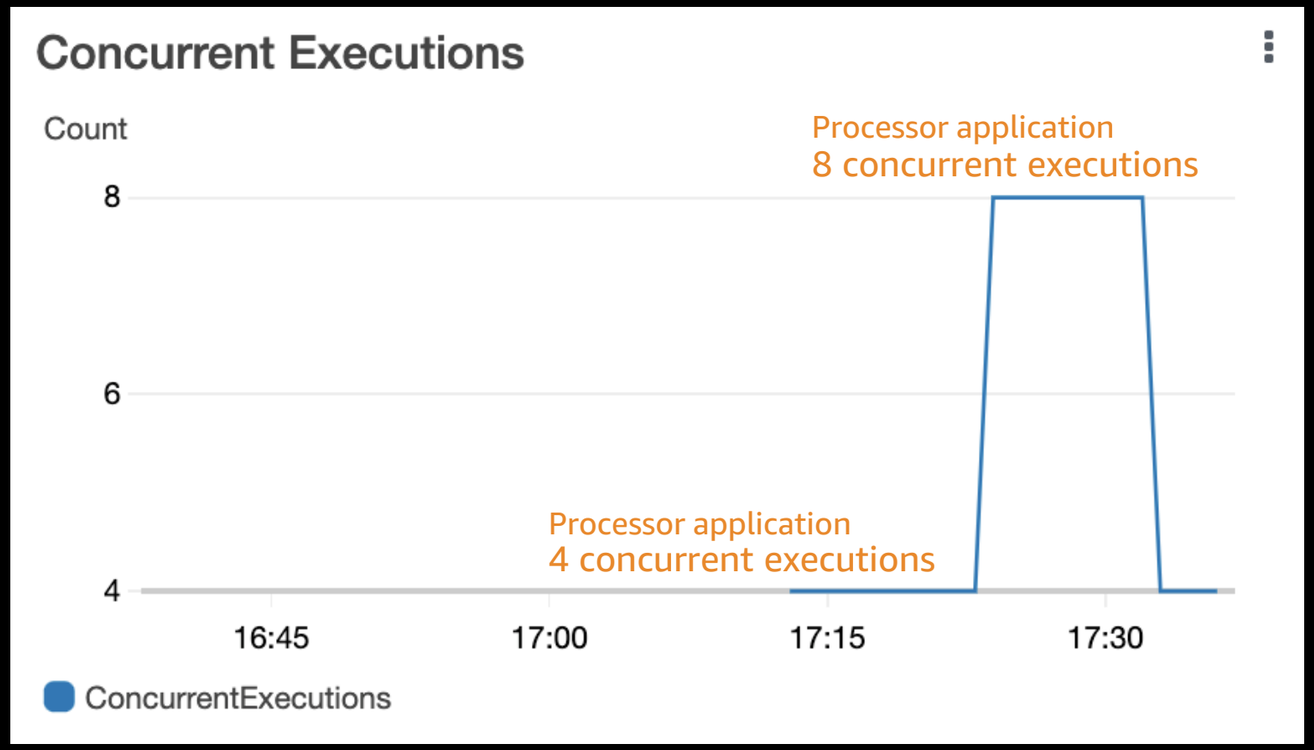
Show me!



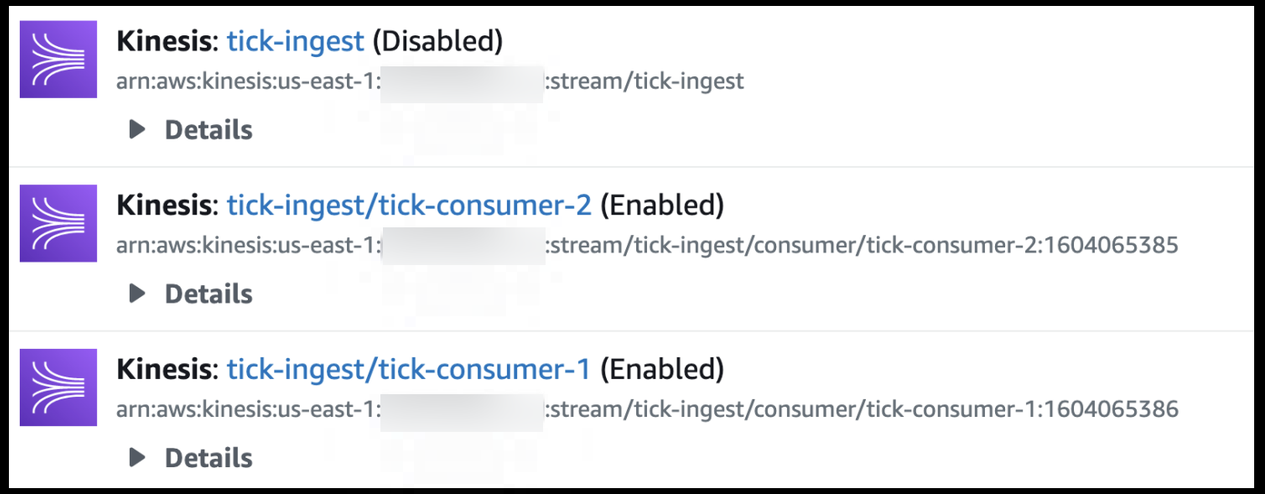


Concurrent batches = parallelization factor

Shards	Consumer Type	Parallelization Factor	Concurrent Executions
4	Standard	1	4
4	Standard	2	8



Show me!



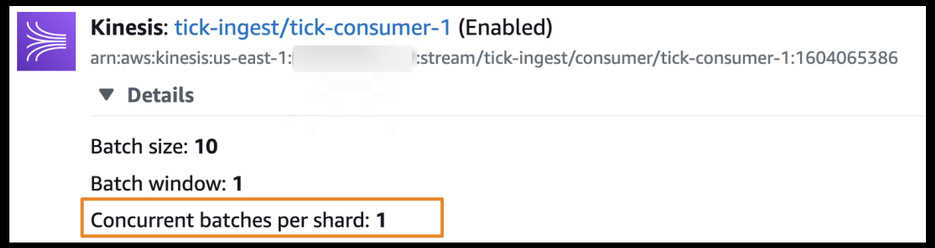
Stream configured with 4 shards

Standard consumer (disabled)

EFO consumer 1 (enabled)

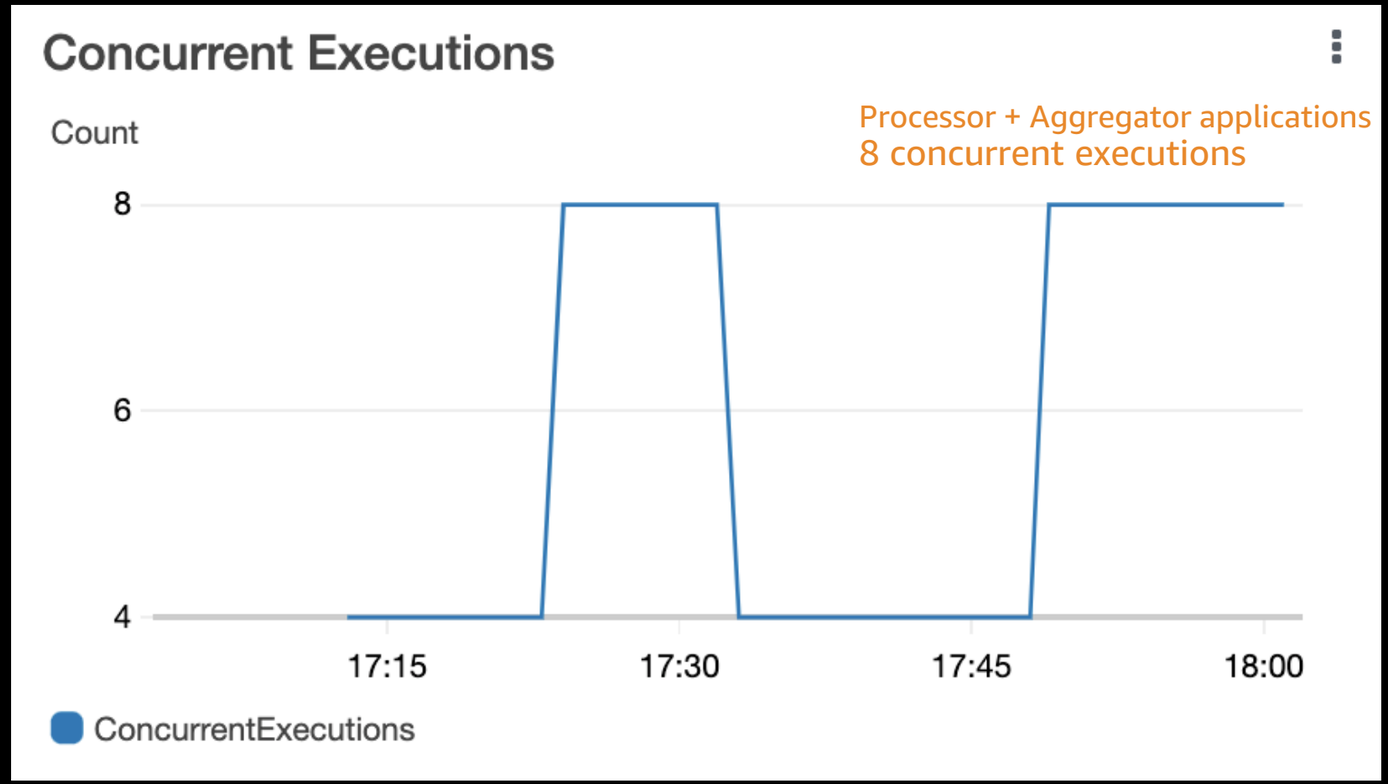
EFO consumer 2 (enabled)

Show me!





Shards	Consumer Type	Consumer Count	Parallelization Factor	Concurrent Executions
4	Standard	Shared	1	4
4	EFO	2	1	8



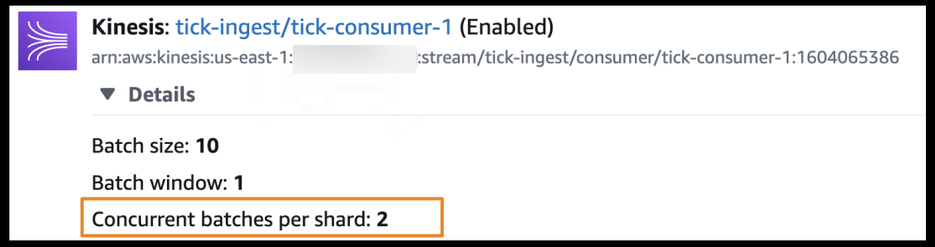
Switched from standard to EFO consumer

Set PF = 1

Show me!

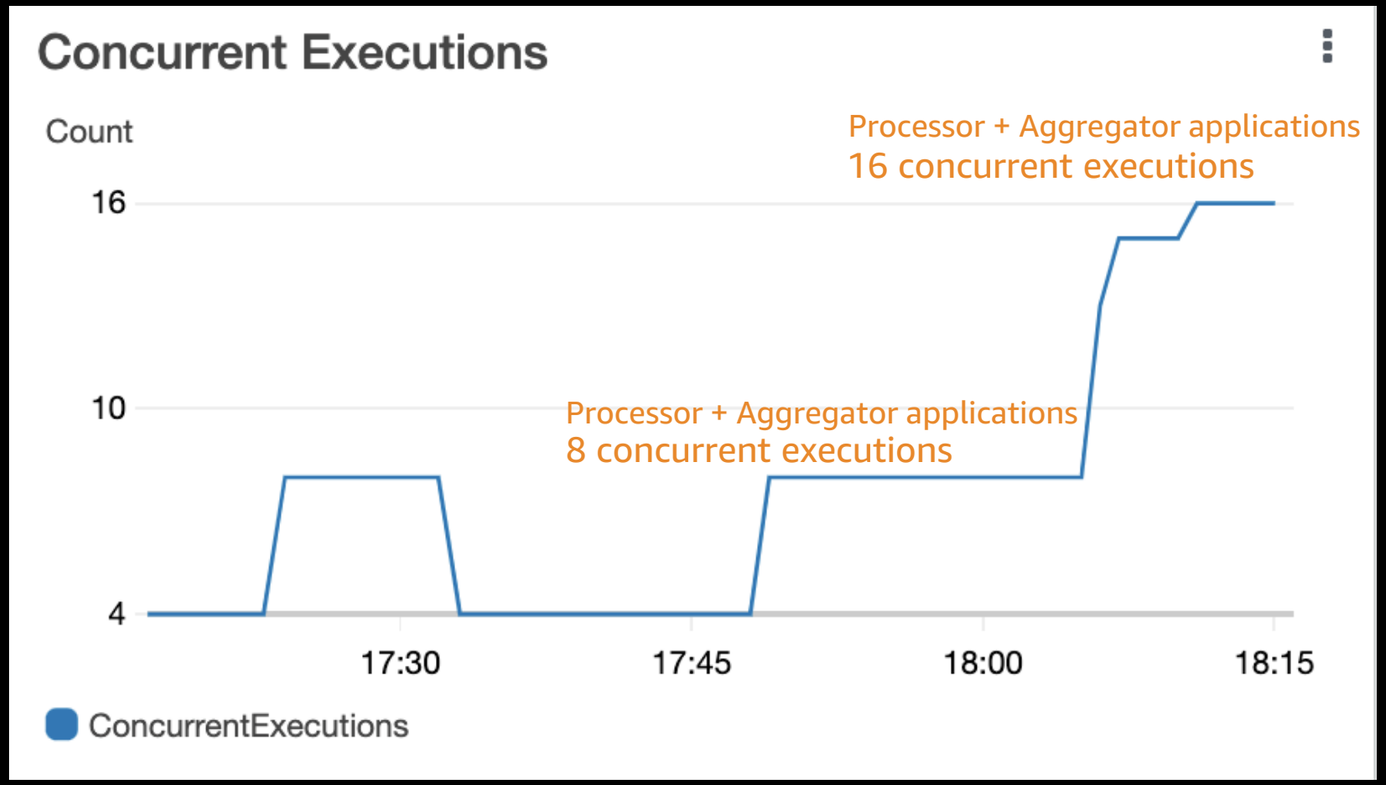
Set PF = 2

Needed to push more records to get full 16





Shards	Consumer Type	Consumer Count	Parallelization Factor	Concurrent Executions
4	Standard	Shared	1	4
4	EFO	2	1	8
4	EFO	2	2	16



Check out these other sessions

SVS303: Scalable serverless event-driven architectures with SNS, SQS & Lambda

SVS304: Understanding AWS Lambda streaming events

Thank you!



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.