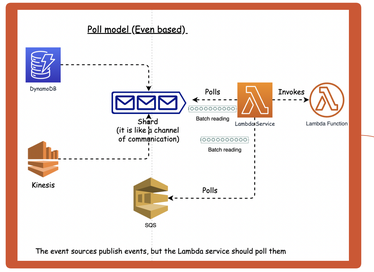
Event Polling

Even Sources

- SQS
  - Queue settings:
    - Standard/FIFO
    - Usually (6 x of function time out)
    - Visibility Timeout
    - Re-drive policy
      - The redrive allow policy specifies the dead-letter queue and max receive count
- Kinesis
  - Stream/Shard settings:
    - number of shards
- DynamoDB
  - Stream/shards:
    - DynamoDB sets the number of shards based on WCUs and RCUs

The lambda service polls the messages from queue or stream . This is "synchronous" call



Polling by Lambda service

- The event sources "push" the events into the queue or stream
- The "lambda service" periodically polls the events from the queue or stream
- Every poll gets a "batch" of events from queue or stream

We have some control in configuring the lambda that is polling queue and streams such as:

- Function timeout
- Function batch size (how many messages to pick up per batch)- from 1 to 10
  - Larger batch size reduce the cost of polling and improve efficiency for the workloads that are processed fast
  - For longer running functions it's better we set "lower batch size" to make sure all the messages are processed before function times out

Not setting the correct batch size causes unsuccessful batch processing

- Design for Idempotency
- Code for partial failure
- Batch Window (max 300s)
  - Increasing Batch Window decreases the the number of polls
  - how long Lambda wait to poll messages from SQS and Stream

Error Handing between Lambda service and event sources

- The lambda service has a built-in mechanism to retry and to get the events from the queue or stream
- See the "Lambda" tab for more info

Events in queue

- Each message is independent
- Rate of getting the messages are different, sometimes we have many sometime none
- Each message is for one consumer
  - Lambda by default starts polling in batches.
    - "5 polling processes"
    - the size of each batch to poll (the # of messages in each batch) is configurable
- Each message in the batch has to be deleted after processing
  - Lambda service does that for you (you do not need to write code for that) if the batch is processes successfully
- Error Handing
  - See the "Lambda" Tab, Error Handing , Polling (event) mode
  - If processing one message in a batch fails, the entire batch is visible again to be retried in the next batch poll
    - If this behavior is not desired, you need to delete the the messages programmatically when the message is a batch is processes
  - Also see how setting a wrong batch size can cause problem in batch processing
- Scaling
  - If the messages queue gets larger, the Lambda adjusts its number of polling processes
  - If there are many error in processing the messages, Lambda scales down, assuming that there is a problem in downstream consumers

Events in streams

- So we need to process them together
  - Some messages are related together and they belong to one core entity
  - Usually we have a high rate of volume of messages
  - Each message can be consumed by more than one consumer
    - So each consumer has its own pointer to keep track of reading from stream
    - Lambda reads the messages in batch from the stream
    - We "do not" delete a message after processing as another consumer may need it again
- Error Handing
  - See the Lambda tab, "Error Handing, Polling (event) mode branch
- Scaling/Performance
  - Lambda will not increase the concurrent invocations unless you increase the number of shards
  - We can use concurrent batches per shard
    - This is called "Parallelization factor"
  - Polling Rate
    - By Default: Lambda reads records from a stream at a fixed cadence (i.e.. once per second for Kinesis data streams) and invokes a function with a batch of records
      - Kinesis
      - Lambda polls each shard 4 times per second.
      - DynamoDB
    - Batch window
      - Batch Window setting in stream allows you to wait as long as 300s to build a batch before invoking a function
      - *This allows less invocation from lambda to stream....that decreases the cost*