Simplified Circuit Breaker Pattern in CMDS

- **Goal**: Design for a error handling mechanism using circuit breaker pattern for CMDS when invoking a remote service or resource.
- **Objectives:**

  Allow Self healing of Remote service, manual intervention
  Mitigate cascading errors from Remote service to CMDS

  Minimization of wastage of the CPU Cycles, Cloud Resources in case of predictable error

- **Status**: Draft
- **Assumptions**:
- **Constraints**:

Problem Context

From CMDS different components are interacting with external services via Rest API calls, SFTP sessions, HTTP resource invocation etc. In these communications, if it encounters frequent long-lasting errors, the immediate retries also tend to fail, makes the situation worse and wastes the cloud resources. Circuit breaker pattern can be applied to prevent the frequent errors by tripping the circuit for a configurable time period and resume the communication only once the issue is resolved.

Error Categorization:

Two broad error categories in different microservices components have been identified - Transient & Intransient.

Transient -

Self-healing failures which are temporary glitches in nature and likely to go away with time. Examples of this error are - cloud errors, throttling error, 5xx server errors, other types of temporary network glitches. This should be retried after certain time to check if the error occurs again.

Examples for Transient Errors - Failure in publishing a message to topic, SQL connection error, S3 bucket error, External API call error, Throttling Error, Rate limiter error, other kinds of intermittent communication error etc.

Intransient -

Aka non-transient, Failures which are permanent and will happen again in similar situation if the cause of the error is not corrected by some external intervention. Requires something more than(manual intervention, reporting) automatic retry.

Examples for Intransient Errors - AWS User Permission Error, DB schema/table/type non-existent error, Data constraint violation error, Data Validation error, Data parsing error, Business error etc.

~~**In general Circuit breaker pattern is applied on Intransient Errors and Retry pattern applies for Transient errors.**~~

In current implementation of Circuit Breaker , only 5xx errors are getting reported and used to activate the Circuit Breaker

Junctions for Circuit Breaker Pattern Application in CMDS

1. **Lambda to Remote API call**
2. **Lambda Function to Remote SFTP server operation**
3. Lambda Function to HTTP Resource Get

Instance Level/ Local Circuit Breaker Library

Using well-known fault tolerance frameworks like Resilience4J enables implementation of patterns like retry, circuit breaker etc on the microservice/lambda instance level. Below examples show how these can be implemented. With these libraries, generally developers only have to decorate or annotate or secure call method to make all communication to that service being wrapped by the circuit breaker logic.

Microservice : Here if the processing fails(due to some intransient error) inside an instance of a microservice, and we know that the error will happen again unless solved explicitly, the local circuit breaker can be used to cut off the connection between the that instance and the message source. Maybe JMS listener can be disabled for implementing that.

Also, retry mechanism can be applied in case of transient errors which also will work on instance level.

Example circuit Breaker

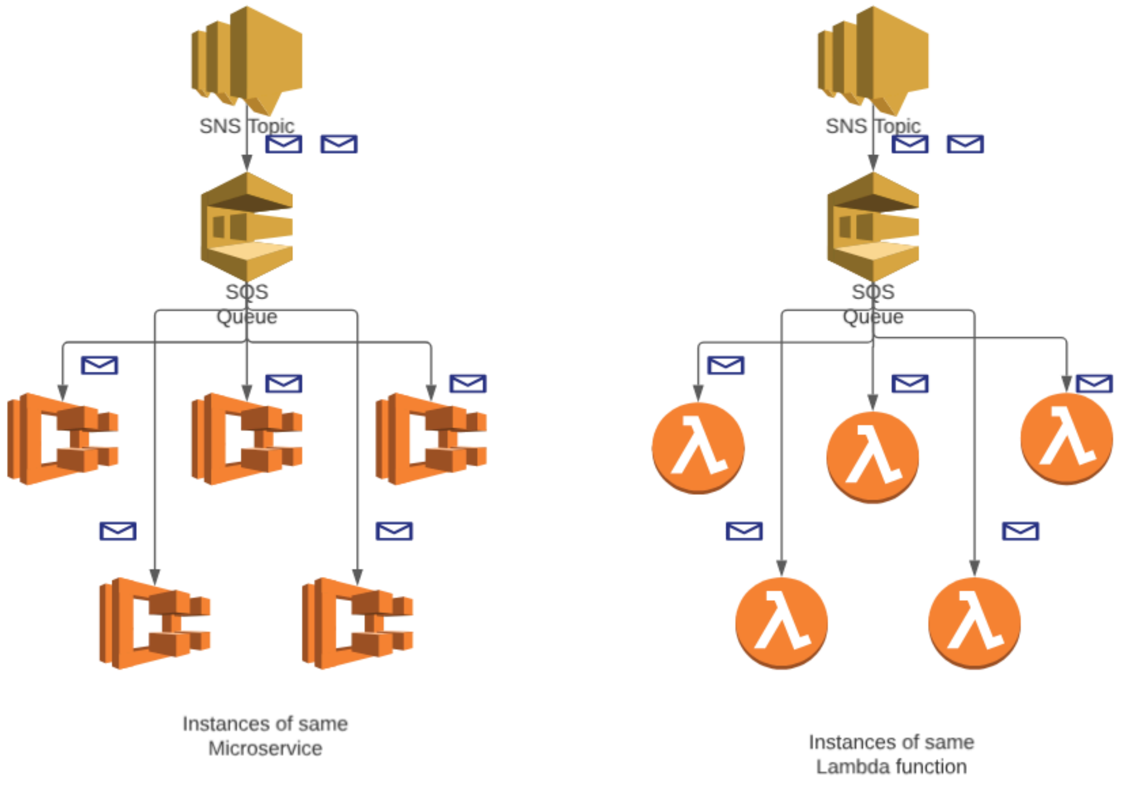
```
@CircuitBreaker(requestVolumeThreshold = 10, failureRatio = 0.5, delay = 1000, successThreshold = 2)
@Retry(name = "throwingException")
public void process(final BaseEvent<? extends BaseHeader> event) throws ProcessingException {
// Service Event Processing Code goes here
}
```

Lambda Function : In case of lambda function this local circuit breaker application is difficult because lambda instances are short living by nature and there is not much point in cutting off the circuit between a message queue and a particular instance of the lambda function. However, retry pattern can be applied to some extent.

Example circuit Breaker

```
@Retry(name = "throwingException")
public void handleRequest(final SQSEvent event, final Context context) // Lambda Handler method
      throws JsonProcessingException, TokenNotReceivedException, InvalidClientException {
 // Service Event Processing Code goes here
}
```

Limitation of Instance Level Circuit Breaker


SNS Topic
SQS Queue
Instances of same Microservice
SNS Topic
SQS Queue
Instances of same Lambda function

However, these frameworks work on each service instance separately and in a cloud-native elastic distributed architecture does NOT serve the purpose well. Auto scaling of clusters, services, lambda function instances are managed by the cloud provider where fault tolerant system instances working in silo does not solve all the problems.

A centralized monitoring and circuit breaker system at the platform level is necessary which will have the runtime view of all the running instances collectively. Thus it would be able to determine the correct course of action based on all those information/data from all the instances of a service/lambda. This can be tackled only at platform level.

Triggering Policy

The triggering or tripping of a circuit happens on the basis of a set of configurable criteria/rules called a Policy. This Policy can be specified generally at platform level OR at microservice level OR at event level. Example policy can include -

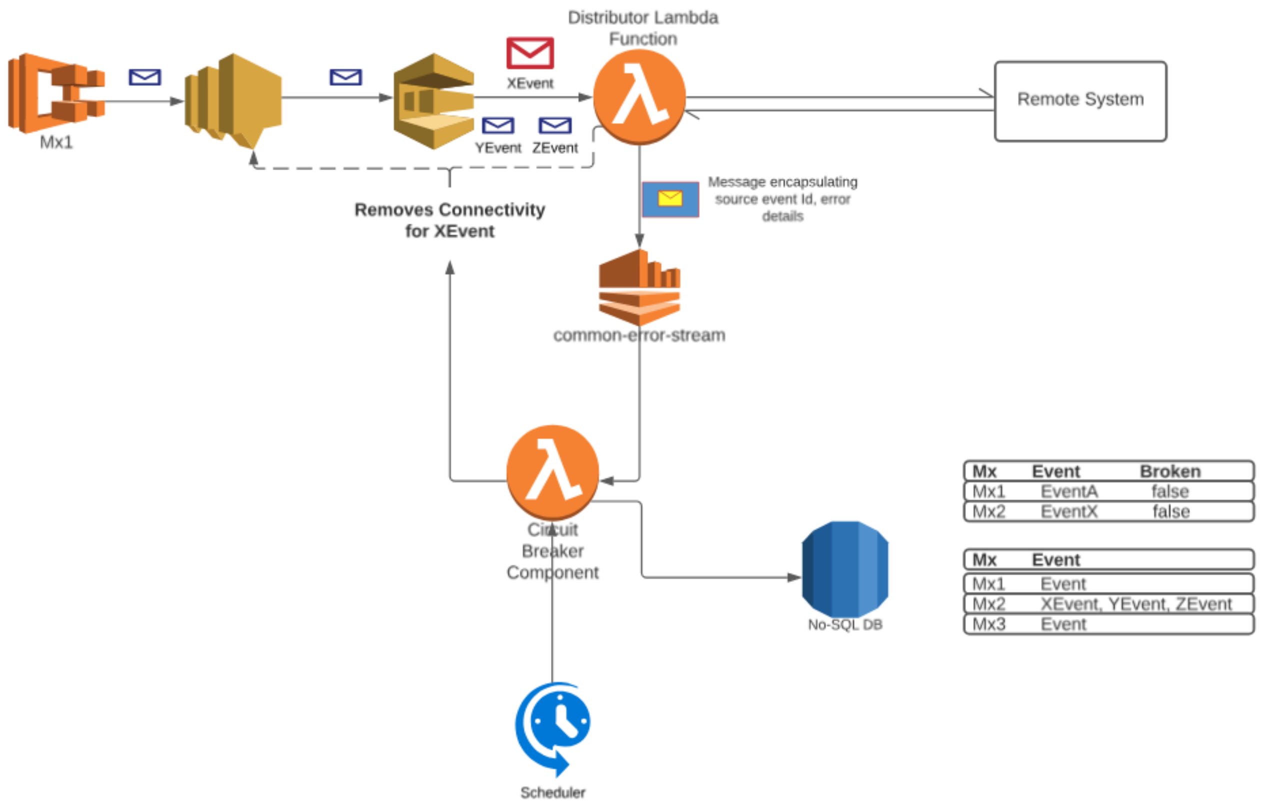
1. We want the circuit breaker to open if 70% of the requests in the last 10s failed
2. We want the circuit breaker to close if 5 simultaneous requests are successful.
3. We want the circuit breaker to open when more than 50% of the recorded calls took longer than 10 seconds to execute.

High Level Solution Design :

New Components Required:

1. common-error-stream Real Time Data Stream - Kinesis
2. Circuit Breaker Service
3. Error Segregator Lib
4. circuit-breaker reference data, events metadata store - Nosql DB like Dynamo DB
5. Infra Script to Implement Open/Close Circuit

Step 1 : Opening the Circuit---


Distributor Lambda Function
XEvent
YEvent
ZEvent
Mx1
Remote System
Removes Connectivity for XEvent
Message encapsulating source event Id, error details
common-error-stream
Circuit Breaker Component
No-SQL DB
Scheduler

Mx	Event	Broken
Mx1	EventA	false
Mx2	EventX	false

Mx	Event
Mx1	Event
Mx2	XEvent, YEvent, ZEvent
Mx3	Event

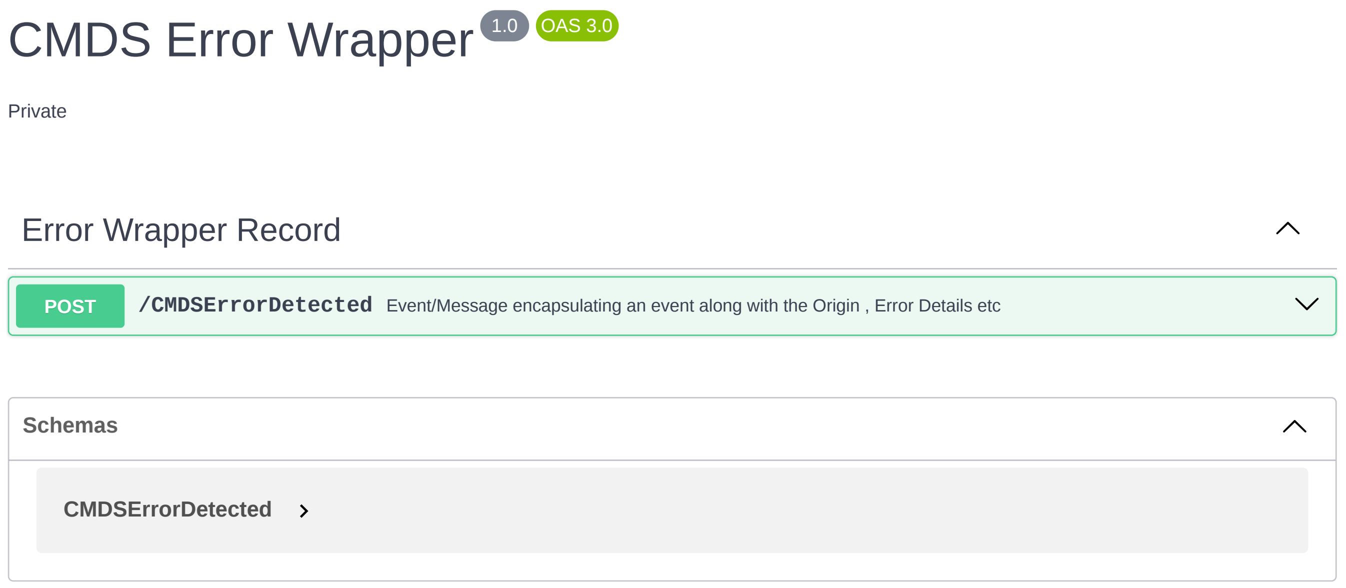
Work Flow :

1. **Dist-Lambda is consuming XEvent, YEvent, ZEvent etc from its topic/queue pair.**
2. **Dist-Lambda encounters Unexpected Error/Intransient Error while calling a remote service/API endpoint/operation .**
3. **Dist-Lambda publishes a message in the common-error-stream Kinesis stream which contains the actual failed message and detailed reason of failure.**
4. Circuit Breaker Service Lambda Function consumes it.
5. **With the help of error segregator module determines if this is a transient/intransient error.**
6. **CB service checks its DB/Cache for checking if this event has reached the threshold of tolerance by checking counts.**
7. **CB service temporarily stores the XEvent; marks the state in DB,**
8. **Tripper service disconnects the Consumer Mx2 from the Topic/Queue for XEvent. This is probably done by modifying subscription Filter of the topic by invoking an Infra script.**
9. **CB Service schedules a check after x minutes for this Event-Mx combination.**
10. **After x minutes CB service sends the previously stored XEvent directly to the relevant queue for testing.**
11. **If the operations team finds the error to be resolved by some other means, there will be provision to manually trip the circuit.**
12. **If Dist-Lambda successfully processes this time it will publish a success message to common-error kinesis stream.**
13. **Once CB service receives success message it updates DB to close the circuit and re-establishes the connection between Dist-Lambda and topic/queue for XEvent.**

Responsibility of Services :

1. **Microservice is responsible for detection of error and for putting a record as a wrapper of message for which the error occurred to kinesis stream. This will be done by decorating the microservice/lambda with some reporter annotation.**
2. **Segregator Service is to determine the error/fault type.**
3. **Circuit Breaker Service will contain the strategy of retry based on factors like error type, frequency, error context etc.**
4. **Tripper service opens / closes the circuit as directed.**
5. DynamoDB will store the error message contents, metadata , context info AND status of each microservice-eventtype-topic combination.

Error Wrapper Structure :



CMDS Error Wrapper

1.0 OAS 3.0

Private

Error Wrapper Record

POST `/CMDSErrorDetected` Event/Message encapsulating an event along with the Origin , Error Details etc

Schemas

CMDSErrorDetected