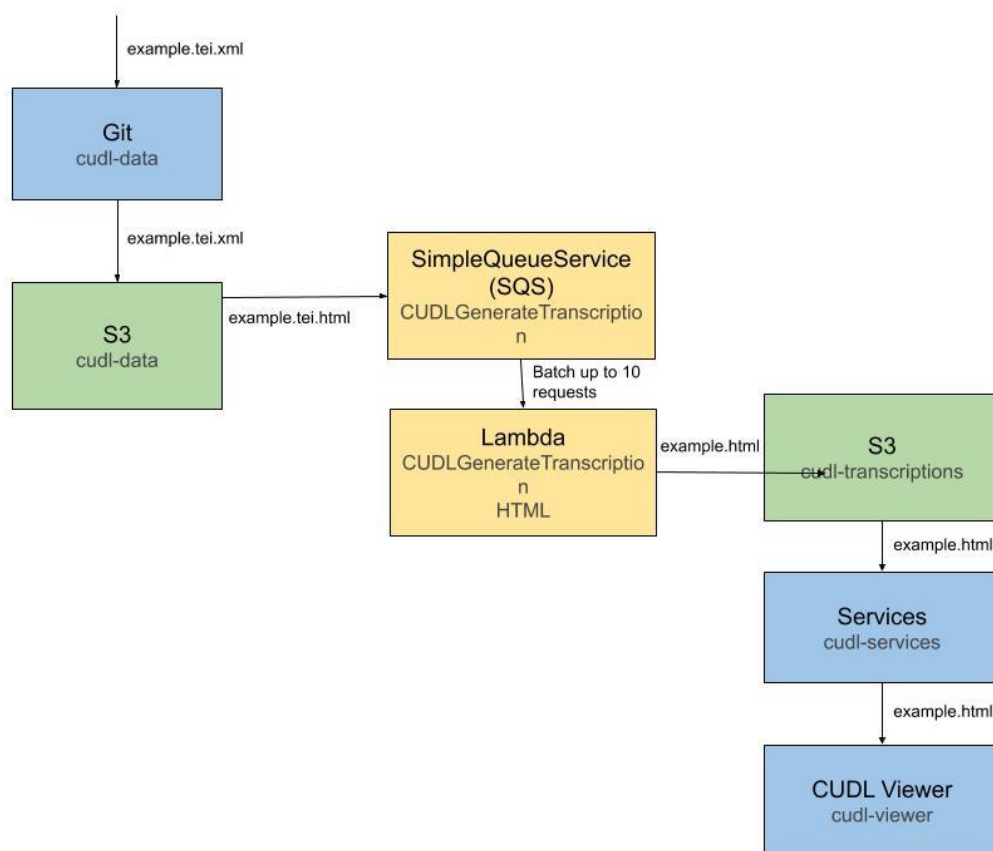


TEI to Transcription Processing



This process automatically generates transcriptions from TEI data committed to the cudl-data bitbucket git repository. These transcriptions should become available a few minutes later under the cudl-staging website.

What happens when a commit is made:

- A commit is made to cudl-data (master)

<https://bitbucket.org/CUDL/cudl-data/src/master/>

- The bitbucket pipeline is triggered which copies data to the S3 bucket 'cudl-data':

<https://s3.console.aws.amazon.com/s3/buckets/cudl-data?region=eu-west-1&tab=objects>

- Any objects in this bucket that are changed trigger an event to be sent to the SQS (Simple Queue Service) Event Queue. You can see the trigger configuration under the s3 bucket properties as follows:

This shows that any create object event and any delete object event will be passed to the queue. There is one event for every object that is changed. So if a commit changes 100 files, there would be 100 events triggered.

Event notifications (2)					
Send a notification when specific events occur in your bucket. Learn more					
	Name	Event types	Filters	Destination type	Destination
<input type="checkbox"/>	CUDLGenerateTranscription	All object create events	data/tei_*.xml	SQS queue	CUDLGenerateTranscriptionQueue
<input type="checkbox"/>	CUDLGenerateTranscription_Delete	All object delete events	data/tei_*.xml	SQS queue	CUDLGenerateTranscriptionQueue

- Under the queue interface you can see two queues are setup:

Queues (2)									
Search queues by prefix									
	Name	Type	Created	Messages available	Messages in flight	Encryption	Content-based deduplication		
<input type="radio"/>	CUDLGenerateTranscriptionQueue	Standard	23/10/2020, 11:23:46	0	0	-	-		
<input type="radio"/>	CUDLGenerateTranscription_DeadLetterQueue	Standard	26/10/2020, 19:02:43	0	0	-	-		

The queue these events are placed in is called 'CUDLGenerateTranscriptionQueue'. This receives the events, and batches them up (if required) and then sends them to the Lambda function for processing. If there is an error processing the queue will try again, by sending the event back after a delay, a set number of times and then it adds the event to the dead letter queue. This is called 'CUDLGenerateTranscription_DeadLetterQueue'. Where it is currently kept for a few days before being deleted.

Lambda triggers (1)				
Search triggers				
UUID	ARN	Status	Last modified	
8d45e6f0-571b-4ee3-a43b-530ebc0b0c07	arn:aws:lambda:eu-west-1:247242244017:function:AWSLambda_CUDLGenerateTranscriptionHTML_AddEvent:LIVE	Enabled	14/11/2020, 15:31:58	

You can see under the queue Lambda Triggers section the function it send the events to. This is called '**AWSLambda_CUDLGenerateTranscriptionHTML_AddEvent**' and the part after the ':' is the **alias** for that function. We want the queue to only send events to the **LIVE** version of the function, as this is the tested version we know works as expected. See below for an explanation on how new versions of the lambda function are tested and released.

What does an event look like?

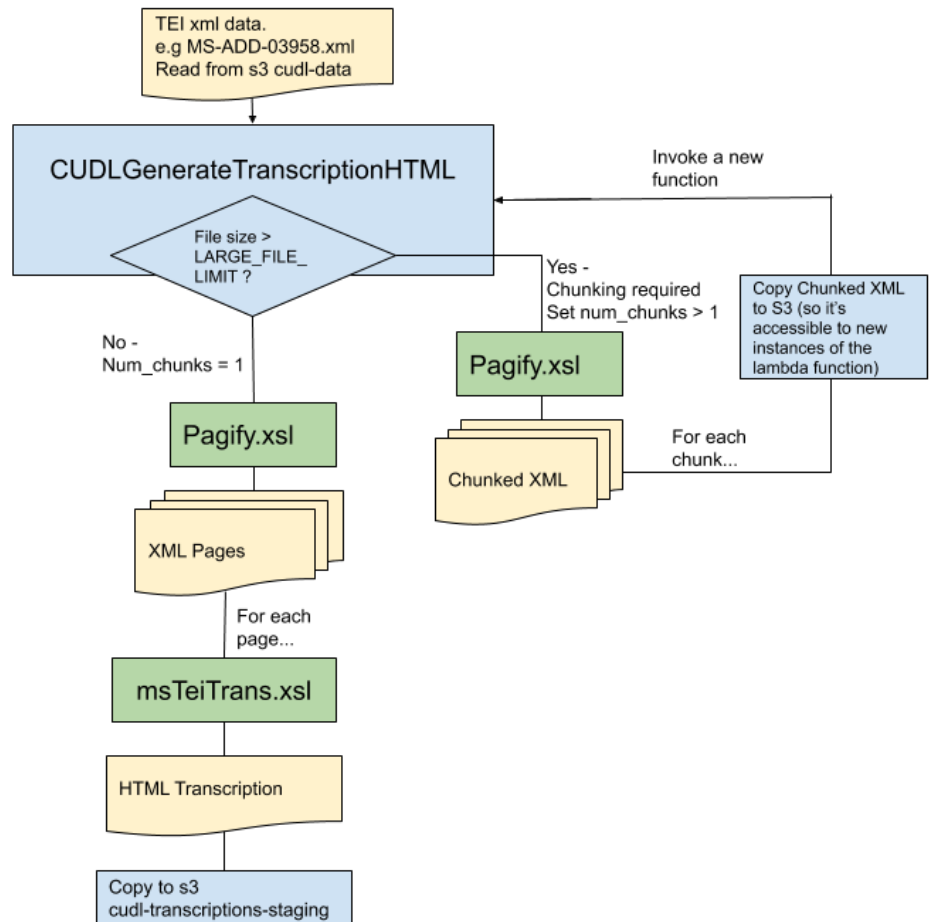
Events are JSON objects, an SQS event contains within it a number of SQSMessages (which is the batch functionality). There is a maximum of 10 messages. Each message corresponds to one event from S3. Within each message the body element contains the original JSON for the S3 event (Create or Delete event) from cudl-data.

For example, an s3 event looks like:

```
{
  "Records": [
    {
      "eventVersion": "2.1",
      "eventSource": "aws:s3",
      "awsRegion": "eu-west-1",
      "eventTime": "2020-11-18T08:55:00.957Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "AWS:AIDATTEGJH6Y2QMKG3XUR"
      },
      "requestParameters": {
        "sourceIPAddress": "35.171.175.212"
      },
      "responseElements": {
        "x-amz-request-id": "045F39CA771367B6",
        "x-amz-id-2": "fMuJx7C2pOGgKiuSAno/ivFq+RFDxsvIzaGSS0ZBI+8bW/zOnsZ65QvX+ELw4VT0K0jslXZ3ZT7CRMImju0A5fzuapW6g9QP"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "CUDLGenerateTranscription",
        "bucket": {
          "name": "cudl-data",
          "ownerIdentity": {
            "principalId": "ATML35UCSMLB4"
          },
          "arn": "arn:aws:s3:::cudl-data"
        },
        "object": {
          "key": "data/tei/MS-ORCS-00004-00005/MS-ORCS-00004-00005.xml",
          "size": 8981,
          "eTag": "796bec1837a6a53829a83198808aab21",
          "sequencer": "005FB4E16A31816465"
        }
      }
    }
  ]
}
```

The information we are interested in is the type of event (create or delete) source s3 bucket and the object key.

What does the Lambda Function Do?



The code for the lambda function is at:

<https://bitbucket.org/CUDL/transcription-lambda-transform/src/master/>

For large files data is chunked because otherwise the limit of 15 mins runtime for a lambda function is reached. The pagify.xsl and msTeiTrans.xsl functions are used to convert the data into a set of html files.

Note: If the number of chunks produced by the chunking process is only 1 then the content is not pagified and cannot be chunked so it is processed as a single chunk (without spawning a separate process).

Bulk Processing

There are cases when all the data might need to be re-processed, for example changing the xslts, in which case we may want to do that locally and upload the contents to s3 directly.

There is an ant build file within the project that allows you to do this. See the README within the project for details.

Once the files are generated you can upload them directly to the cudl-transcriptions-staging bucket using the aws cli software.

See s3 sync command here:

<https://docs.aws.amazon.com/cli/latest/userguide/cli-services-s3-commands.html#using-s3-commands-managing-objects-sync>

E.g. 'aws s3 sync <local path> <s3_path> --size-only

You can run with --dryrun to test what will be copied before running the command.

NOTE: Make sure the paths you are uploading to match your src file directory structure so you don't duplicate files.

Releases for the data:

Once the data is ready for release you can release it in the normal way, this will cause the data to be copied to the 'live' branch of the cudl-data git repository. Doing this will cause the bitbucket pipeline to run and sync data from cudl-transcriptions-staging to cudl-transcriptions-live.

This means no additional steps are needed to release the transcription data.

Releases for the code:

When the code is committed to the master branch of '[transcription-lambda-transform](#)' it will cause a bitbucket pipeline to run. This does two things:

- It deploys the latest version of any web resources the transcriptions need to cudl-transcriptions-dev s3.
- It deploys the latest version of the lambda function code and tags a new version for it with the \$LATEST alias. This sets the ENV variable to 'DEV' making the function use the cudl-transcriptions-dev bucket so it can be tested without impacting anything else.

Once the code is tested it can be released to live by manually triggering the pipeline 'Publish to Live' from bitbucket which will tag the function with the LIVE alias and copy the web resources to the staging and live buckets.

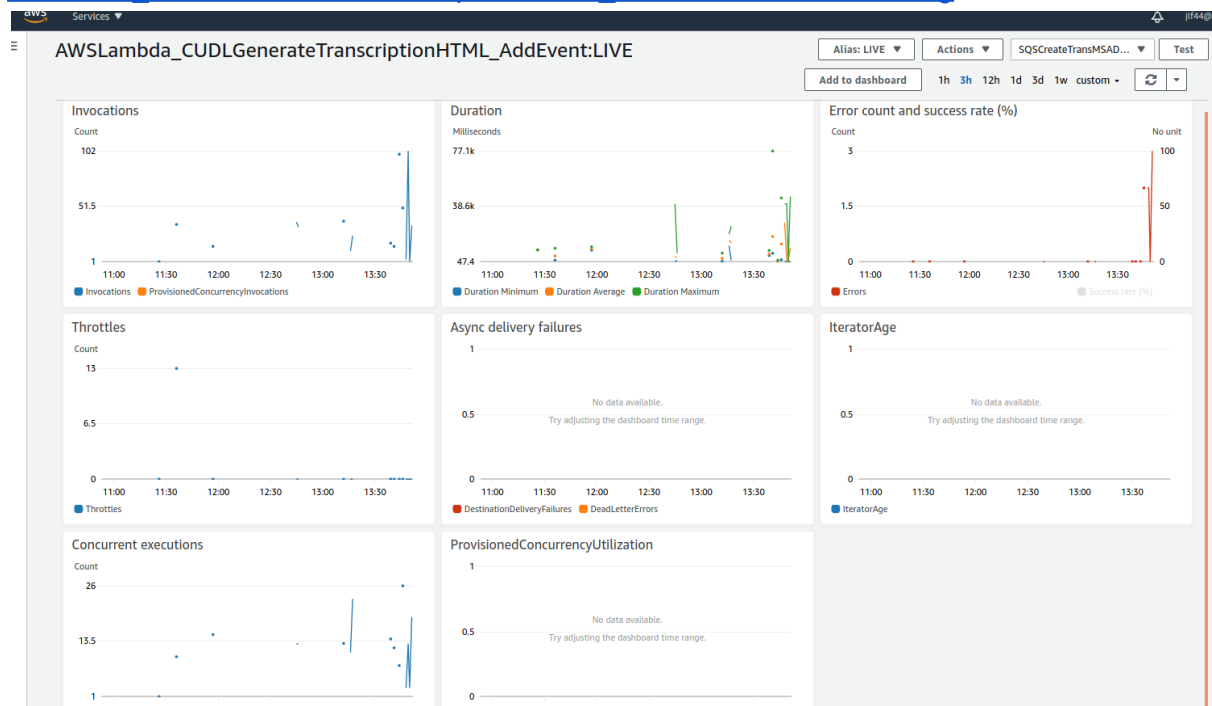
This means that the function that's triggered with any data changes should be the LIVE version, but the DEV version is available for testing using data from cudl-transcriptions-dev. Once happy with the new version, run the bitbucket pipeline to make this version LIVE.

Concurrency

By default lambda has a limit for 1000 concurrent invocations (our account limit). There may be circumstances (when testing etc) where you want to limit this which can be done on the lambda configuration by assigning reserved concurrency for the function as below:

You can see graphs for the recent invocations of the lambda function at the monitoring tab, e.g.:

https://eu-west-1.console.aws.amazon.com/lambda/home?region=eu-west-1#/functions/AWSLambda_CUDLGenerateTranscriptionHTML_AddEvent?tab=monitoring



The screenshot displays the AWS Lambda console CloudWatch Logs Insights tab for the function `AWSLambda_CUDLGenerateTranscriptionHTML_AddEvent`. The table shows recent invocations and most expensive invocations.

Recent Invocations

#	Timestamp	RequestID	LogStream	DurationInMS	BilledDurationInMS	MemorySetIn...	MemoryUsedInMB
1	2020-12-07T11:37:00.398Z	757f03de-f274-5d5e-a9be-5b96e46dd790	2020/12/07/[42]bc88607785ef4088b583b9dd10495db8	2488.19	2489	512	240
2	2020-12-07T11:36:59.519Z	776394fb-d944-5b05-a3e8-cb3c5f5c2f93	2020/12/07/[42]bdc8aef18f47947df93ad3b2197ecbcd	1696.99	1697	512	242
3	2020-12-07T11:36:59.506Z	c26246d1-f6dc-519e-adf5-1e99ecb3656	2020/12/07/[42]bdc00685388b4b8780d3d791be51d751	1952.6	1953	512	243
4	2020-12-07T11:36:58.912Z	19ac3deb-55a5-58b9-802f-65a343b28669	2020/12/07/[42]j909bc02392b4e2abff7698512bc9aba	3618.75	3619	512	240
5	2020-12-07T11:36:58.731Z	8b0bd664-9961-5486-803c-8e2c8f767a56	2020/12/07/[42]j3a6ad33354745cab82f13878b1c26cf	2843.67	2844	512	243
6	2020-12-07T11:36:57.969Z	8e2885a4-eb2a-5d5f-b7d2-ddecf43c29c9	2020/12/07/[42]e2930b86b6fd43908959b25e556940de	2991.31	2992	512	242
7	2020-12-07T11:36:57.680Z	4a1c342a-54a3-5329-b76a-6c62424b7bf	2020/12/07/[42]bdc8aef18f47947df93ad3b2197ecbcd	1905.41	1906	512	242
8	2020-12-07T11:36:57.489Z	60afa136-5699-5e89-90be-f97e408e1059	2020/12/07/[42]bdc00685388b4b8780d3d791be51d751	1817.7	1818	512	243
9	2020-12-07T11:36:57.076Z	94de4e0b-b78d-5fb3-9d0e-1643a19a4833	2020/12/07/[42]bc88607785ef4088b583b9dd10495db8	2021.39	2022	512	234

Most expensive invocations in GB-seconds (memory assigned * billed duration)

#	Timestamp	RequestID	LogStream	BilledDurationInMS	MemorySetIn...	BilledDurationInGBSeconds
1	2020-12-07T11:36:49.409Z	b94e888d-a31b-53cf-9b69-40c5b6e8ed1f	2020/12/07/[42]j3a6ad33354745cab82f13878b1c26cf	9423	512	4.7115
2	2020-12-07T11:36:48.602Z	6156c011-54e9-5e0d-9a2b-985941127ecc	2020/12/07/[42]b5056f6e2c444c2c8bdc3035c274ab343	9375	512	4.6875
3	2020-12-07T11:36:48.628Z	39f796fc-2fa5-55eb-adee-ceb43a530825	2020/12/07/[42]bdc00685388b4b8780d3d791be51d751	9161	512	4.5805
4	2020-12-07T11:36:48.660Z	b496642b-8237-5392-b6bd-3c6397cac95c	2020/12/07/[42]e2930b86b6fd43908959b25e556940de	9107	512	4.5535
5	2020-12-07T11:36:48.464Z	ea64ea26-9b9f-57a1-bb1b-b26e8bc026971	2020/12/07/[42]j3a6ad33354745cab82f13878b1c26cf	9039	512	4.5195
6	2020-12-07T11:36:48.399Z	aeb0ea27-26cb-52a2-bc8b-b26f3355f02	2020/12/07/[42]bdc8aef18f47947df93ad3b2197ecbcd	8708	512	4.354
7	2020-12-07T11:36:48.051Z	f69674db-e6d3-50b8-83bb-3f00f32f544e	2020/12/07/[42]j909bc02392b4e2abff7698512bc9aba	8634	512	4.317
8	2020-12-07T11:36:47.654Z	eddc7b1a-8625-50b8-800a-235293d85a76	2020/12/07/[42]j909bc02392b4e2abff7698512bc9aba	8352	512	4.176
9	2020-12-07T11:26:40.329Z	f7fe40ac-7dce-5e33-8bbd-18e46e568b32	2020/12/07/[42]j3a6ad33354745cab82f13878b1c26cf	8278	512	4.139

You can find logs at cloudwatch under:

[https://eu-west-1.console.aws.amazon.com/cloudwatch/home?region=eu-west-1#/logsV2:log-groups/log-group/\\$252Faws\\$252Flambda\\$252FAWSLambda_CUDLGenerateTranscriptionHTML_AddEvent](https://eu-west-1.console.aws.amazon.com/cloudwatch/home?region=eu-west-1#/logsV2:log-groups/log-group/$252Faws$252Flambda$252FAWSLambda_CUDLGenerateTranscriptionHTML_AddEvent)

These show the output of the latest function runs, including intermediate files generated etc. If there are any errors you can debug the causes here. Remember that chunked data will cause separate invocations so will have separate logs.

You can specifically look in the Log Insights section and use the saved query to find the log files for any errors:

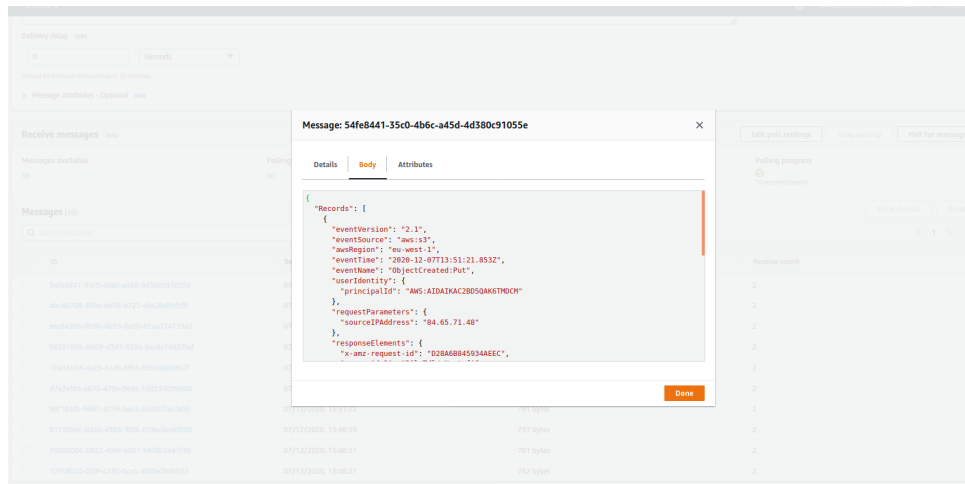
The screenshot shows the AWS CloudWatch Logs Insights console. The left sidebar contains navigation links for CloudWatch, Alarms, Billing, Logs, Metrics, Events, ServiceLens, Container Insights, Performance Monitoring, and Synthetics. The main panel displays a query for the log group `/aws/lambda/AWSLambda_CUDLGenerateTranscriptionHTML_AddEvent` with the query `fields @message | filter @message like /Exception/`. The results show 9 records, all indicating a timeout waiting for connection from the pool. The right sidebar shows a list of saved queries, including `List_Errors_CUDLGenerateTranscriptionHTML`.

You can view individual messages in a queue by going to the SQS Queue and selecting “Send and Receive Messages”, then “Poll for messages”.

The screenshot shows the Amazon SQS console for the queue `CUDLGenerateTranscription_QueueLetterQueue`. The 'Send and receive messages' page is active. The 'Send message' section shows a message body input field and a delivery delay of 0 seconds. The 'Receive messages' section shows 56 messages available, a polling duration of 30 seconds, and a maximum message count of 10. The 'Poll for messages' button is highlighted. Below, a table lists the received messages with columns for ID, Sent, Size, and Receive count.

ID	Sent	Size	Receive count
54fe0841-350b-4b6c-a45d-4d300c31055e	07/12/2020, 15:51:24	782 bytes	2
abc4670b-50ba-4a56-b725-cde2bdfc9f9f	07/12/2020, 15:51:22	782 bytes	2
eeed4395-d89b-4659-8a0b-81a0374733a2	07/12/2020, 15:51:22	782 bytes	2

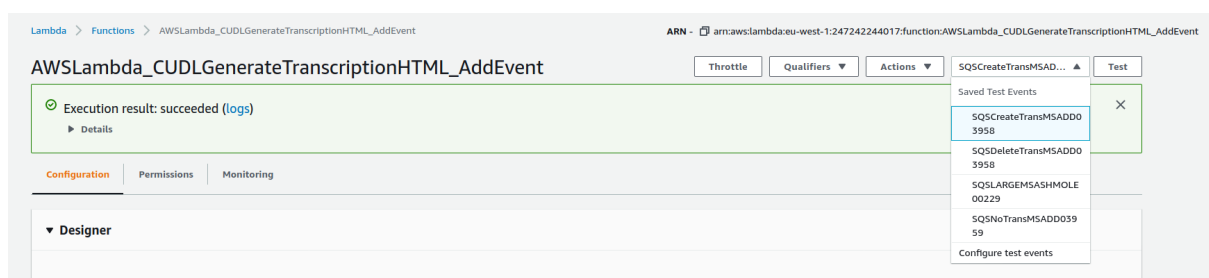
And you can view the body of that message:



Lambda: Tests

The function itself has tests that are configured under the configurations tab:

https://eu-west-1.console.aws.amazon.com/lambda/home?region=eu-west-1#/functions/AWSLambda_CUDLGenerateTranscriptionHTML_AddEvent?tab=configuration



This is setup with some sample events for data with and without transcriptions and for data that needs to be chunked or not. You can see the result in the cloudwatch logs and processed files will be under the "html/test" directory on cudl-transcriptions-dev. Source data is taken from the s3 data directory under the "test" folder.

This means tests can use a fixed set of data that's separate to the live data set.