PLURALSIGHT | HELP CENTER

Lambda troubleshooting

Tags: ACG

Background

Lambda debugging is not significantly different from any debugging process. You need to have a mental model of how the system and application works, and look for the clues that narrow down the cause of the issues.

All of our labs, and the majority of new internet and in-house development are now using browser-based or API-based interconnects so I will discuss this in terms of an HTML browser environment. If you are doing something different, the elements will be similar but the tools may be different.

You need to have a mental model of how the stack works.

- You interact with a browser which runs HTML code to render (paint) the web page in your browser window.
    - HTML can do a lot of things, however more dynamic capability can be achieved by using the HTML to run Java scripts on your browser. This is called client side scripting and is 100% compatible with S3 so it is a common add-on for Lambda environments. However it is not essential.
- The HTML code calls and paints objects that are either on your local system or downloaded. Each of these calls will result in a success or failure. These are reported by HTML status codes which can be seen in your browser 'Developer Tools / Console'.
- When you are working with Lambda the objects are provided either by a cloud object store (S3), or the API which sends an object or chunk of formatted HTML code back to your browser to make use of.
- While the API Gateway could send results back by it'self, normally it will just proxy for a back-end services. In this case we are focusing on Lambda.
- Running code on Lambda is really just the same as running it on any other server. There are layers upon layers of handles and code and subroutines that are called and the execution is passed up and down these layers to complete the tasks at hand. Some of these layers are in the code you provide, and some are above it in the lambda handler, and some are below in the interpreter or underlying libraries such as C.

When Lambda or some other process is completed, the results are passed back up through each layer and service until your browser paints the results.

As you debug problems keep this hierarchy in mind and mentally place each log and error report you see where in the stack the issues are occurring.

Finding the problem

There are lots of approaches to debugging.

1. The natural instinct is to just go look at things randomly to see if you have made a mistake. For a small project and when you have some basic understanding of why the error occurred this can be effective. Did I add the comma, did I add the correct 'Environment Variable' etc. However if you cannot spot the error in 30

seconds, stop wasting time and start looking systematically.

2. Trace the error hierarchically.
    A. Start at the top (at the browser) and follow the traffic down.
        - Is there an error on the browser?
        - Is there an unexpected failure in the Developer Tools / Console HTML status codes?
        - If something timed out what element does the Developer Tools / Network indicate took too long?
            - Learning to use the Browser Developer Tools to even a very basic level can shorten your debug time. And improve your confidence enormously.

    B. You can track activity through your API Gateway via CloudWatch. This is not enabled by default. Our labs are relatively simple and any problems are unlikely to be in the API, so only do this if you cannot resolve it elsewhere. If you are having a persistent problem then consider enabling logging for a while to see the traffic passing back and forth and any anomalies that may be present at the API.

    > **Note:** The logs can build up so remember to stop it or set up lifecycle management on your logging bucket to age the logs out.

    C. Lambda writes Logs automatically to CloudWatch and this is a great source of information, and where we will focus most of the following discussion.

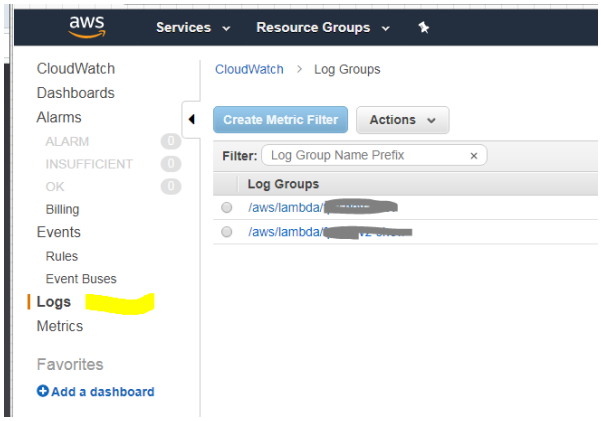
3. Start at the most easily accessed Logs. In most cases this will be the Lambda / CloudWatch logs. However depending on the situation there may be other material that you can review more easily.

Guidelines

A few guidelines I recommend:

1. Use a separate browser for testing the application. (I find Opera and Safari are excellent for testing.)
2. Before each new test:
    A. Point your browser at a neutral page and Clear the Browser cache so that the objects from the last test do not contaminate the next test. (This is less impacting to your other activities if you use a separate browser.)

    B. Clean out the CloudWatch Logs between tests. Delete old logs so that you start from scratch. This will make it easier to see which Lambda functions did and did not run, and reduce the lines you need to read through.



C. Be clear in your mind which Lambda function(s) should run for each activity. Run one test and check the CloudWatch logs to see which Lambda functions did and did not run. Is something missing, is there something extra you did not expect? If things are not as you expected, check your understanding then look at how functions are being called and why it may or may not have run.

3. Watch for typos and look-alike characters: 1 l I o 0 O , . non-display special characters vs. space or tab.

4. Check the Discussion Forums and see what others have experienced, and how they may have resolved it. When you solve your issue don't forget to post advice for those who come after you, or up-vote useful posts.

5. If in doubt, recreate the Resource or Lambda function that is being problematic. It is good practice, and it gets faster each time you do it. Learning from your mistakes can be very powerful, but you also need to manage the time you are investing in small errors.

6. Use a proper code editor for typing and editing code. Don't use Word or Office tools, they have a bad habit of adding formatting and special characters that will mess things up. There are plenty of good quality free editors available. TextWrangler, Atom, PyCharms, SubLime, NotePad++, etc are some of the free and very good products available.

Reading the Lambda / CloudWatch logs

The Logs in CloudWatch may look intimidating when you first see them—so many rows of cryptic information. However with a few simple steps you can make it easier to understand.

If you followed the above guidelines you will have started with a clean environment and within 30 seconds of the function running you should have a set of logs available to review (you may need to refresh your AWS console to make them visible).

Below are some guidelines of what to look for to demystify the logs.

- You will note that the Log is in time-sequence with the last action at the bottom and them getting progressively older as you move or scroll up the list.

- Each line represents an event. For some, the body is a few words, others may be a large blob of text (see Tip below).

- Each line starts with a key word. START, END, REPORT, or a status [INFO], [DEBUG] etc. Also note that START / END / REPORT are a set which describes an execution cycle.

- The REPORT record is a summary of the Lambda job. Quickly check the time and memory use to see if it is what you expected.

- The key words [INFO], [DEBUG] etc are self evident. The [INFO] and [DEBUG] are generally only of interest in

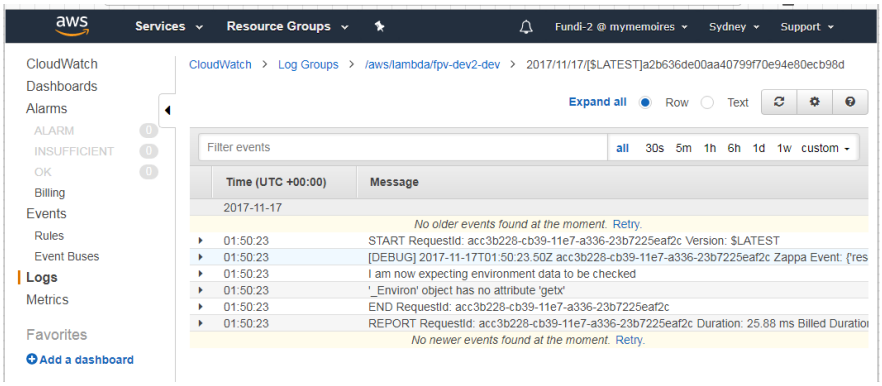
terms of what they inform you about [ERROR]. So start with the [ERROR] entries.

- Recall that the older entries are UP the screen. So when you find a fault read UP to see what precedes it, what was running, what parameters were passed, were there any console messages in the logs showing what the developer was expecting to happen.
- If you add a 'print' or 'console' statement into your code it will be written in the log (see AWS Lambda features (external site, opens in new tab)). You can always add a 'print' or 'console' statement to the code as part of debugging to help you confirm each section of codes are being processed. The time taken to add these can significantly shorten the debugging time.

> **Tip:** When you have a large blob of text, to make it more readable copy it to your code editor and start separating the text into short sections so that each line is a unique piece of information. This will take a few minutes, but will help you understand what it is telling you and if there is anything that looks out of place.

Worked example

So let's look at this error here.



1. The last statement is a `REPORT` at the bottom
   `REPORT RequestId: acc3b228-cb39-11e7-a336-23b7225eaf2c Duration: 25.88 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 46 MB`

   The process took 25.88 milliseconds and 46 MB or memory. So clearly it was not a runaway process.

2. The `START` and `END` statements have matching Request IDs
   `START RequestId: acc3b228-cb39-11e7-a336-23b7225eaf2c Version: $LATEST`

   Therefore I can be certain that everything I need is between these two. Plus the version of the code is `$LATEST` which is useful to know if you have multiple versions on file and there is a chance that the wrong one is being executed.

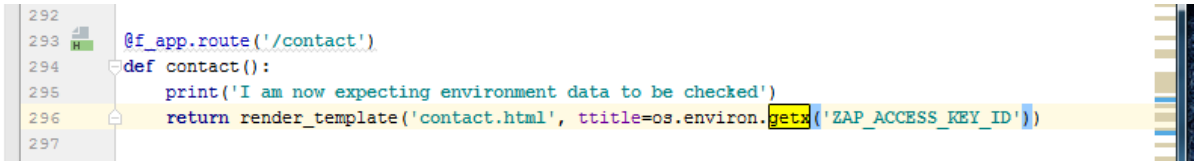
3. There is a `[DEBUG]` which is a big blob of text, but looking closer I can see `'httpMethod': 'GET',` near the beginning so it is most likely the HTML command and parameters that is invoking this function. I will break it down and read it in detail later if I think the problem is caused by a malformed request.

4. The next line is a sentence.
   `I am now expecting environment data to be checked`

I recognize this as a 'print' statement that I put in my code. This will help me localize where the issue happened

5. The final line is somewhat cryptic. `'_Environ' object has no attribute 'getx'`
   Let's read it carefully. `'_Environ' object has no attribute 'getx'`. I know that near where I put the print statement above I was looking for environment variables. So this may be related to environment variables. What is attribute `'getx'`? We know from the statement it is something that `'_Environ'` object does not have, so this could be the source of our error.

   Now we go to the code and look for the print statement and a `'getx'` attribute (whatever that is).



   By searching for `getx` I found one occurrence. It is right below the print statement so that correlates well. I could check the os.environ function, but I can see right away that there is a stray 'x' that does not belong there.

Traceback records

One of the log types that you might see is the Traceback or Stacktrace. This is a report on the event through a hierarchical stack of processes and sub processes.

There are a few things to note when reading these:

> An error occurred (**ResourceNotFoundException**) when calling the PutItem operation: Requested resource not found: ResourceNotFoundException
> **Traceback (most recent call last):**
> File "/var/task/**index.py", line 23, in lambda_handler 'status' : 'PROCESSING'**
> File "/var/runtime/boto3/resources/factory.py", line 520, in do_action response = action(self, *args, **kwargs)
> File "/var/runtime/boto3/resources/action.py", line 83, in __call__response = getattr(parent.meta.client, operation_name)(**params)
> File "/var/runtime/botocore/client.py", line 312, in _api_call return self._make_api_call(operation_name, kwargs)
> File "/var/runtime/botocore/client.py", line 605, in _make_api_call raise error_class(parsed_response, operation_name)
> ResourceNotFoundException: An error occurred (ResourceNotFoundException) when calling the PutItem operation: Requested resource not found

Think about the opening error statement (`ResourceNotFoundException`). This is a big clue that it was looking for something that it could not find. Either your program asked for something that does not exist, or it was expecting you to provide something that you did not provide.

The statement (`most recent call last`) is telling you the order to read the log. The last thing to fail will be at the bottom, and the first thing to fail (closest to the cause) will be near the top. That does not mean that you should not read all of it. The clue you need may be in the middle so read it from bottom to top.

I recommend that you start by trying to look for 'your' program in the log. This is what you have the most control over and at this level of your development is the most likely source of problems. In this case it is one of our course

files `index.py` in `lambda_handler`, and the problem is related to `line 23`, with special mention of the phrase `'status' : 'PROCESSING'`.

Conclusion

- Keep a clean browser for testing so that you can avoid contaminating your current test with old objects.
- Keep a clean log space so that you can identify exactly which logstream to read, and be able to focus on just a half dozen lines of log and very quickly identify the issue
- Don't let the logs intimidate you. Look at the logic and patterns, isolate the area you need to look at and then read the lines one at a time and consider what is being said.
- Read the clues you are given such as Keywords and Summaries about what is happening.
- Don't forget to look at the whole hierarchy from the browser to the lowest process. Don't get tunnel vision and look at just the code you think is the cause or just one log.
- Take 20 minutes and learn the basics of using the Developer Tools in your browser so that you can see what is happening in the browser background.
