Member-only story

Datadog monitors AWS Lambda and lambda logs

Feng Li · Follow
8 min read · Jun 12

Listen Share

Soy bean field next to Seaton Hiking Trail, Pickering ON. June 11 2023

1 Monitor Lambda metrics using Lambda extension

Lambda extension is a feature in AWS. "Extension" works as a specific lambda layer to be added to existing lambda function. Those layers can then be invoked by lambda function to complete some specific tasks. For example Datadog has developed "Datadog lambda extension" to collecting lambda function metrics, logs and pass data back to Datadog services.

To use Datadog Lambda extension, we need to install the extension first and then configure Datadog with AWS account and Lambda integration.

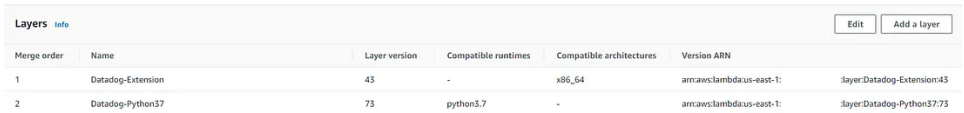
1.1 Installing Datadog lambda extension to your Lambda function in AWS

Base on Datadog doc, following steps are for a Python Lambda function. We'll leverage a Linux host, install Datadog command interface tool, run the tool to install extension to our AWS Lambda function.

So we'll be asked where our Datadog service/account is — need to provide our Datadog API key. Also need to get credentials ready for AWS command line access.

```
#### 0 Setup AWS account credential if it's not done yet
C:\Users\feng\.aws>pwd
/c/Users/feng/.aws

C:\Users\feng\.aws>ls
config  credentials

C:\Users\feng\.aws>cat credentials
[default]
aws_access_key_id = xxxx
aws_secret_access_key = yyyy

#### 1 Install Datadog command interface tool
C:\Users\feng>npm install -g @datadog/datadog-ci

#### 2 Use the tool to install extension in an interactive method
C:\Users\feng>datadog-ci lambda instrument -i

🐶 Instrumenting Lambda function

[!] Configure AWS region.
? Which AWS region (e.g., us-east-1) your Lambda functions are deployed? us-east-1

[!] Configure Datadog settings.
? Select the Datadog site to send data.
Learn more at https://docs.datadoghq.com/getting_started/site/ datadoghq.com
? Which type of Datadog API Key you want to set?
Learn more at https://app.datadoghq.com/organization-settings/api-keys Plain text API Key (Recommended for trial users)

? API Key: bf0xxxc5eb
√ Fetched 16 Lambda functions.

? Select the functions to modify (Press <space> to select, p.s. start typing the name instead of manually scrolling)
custom-errors-in-log-lambda
? Enter a value for the environment variable DD_ENV (recommended) custom-lambda-errors
? Enter a value for the environment variable DD_SERVICE (recommended) custom-lambda-errors
? Enter a value for the environment variable DD_VERSION (recommended) 0.1

[Warning] Couldn't add source code integration, continuing without it. Error: Couldn't get local git status.
√ [us-east-1] Fetched 1 Lambda configurations.


[Warning] Instrument your Lambda functions in a dev or staging environment first. Should the instrumentation result be unsatisfactory, run `uninstrument` with the same arguments to revert the changes.

[!] Functions to be updated:
        - arn:aws:lambda:us-east-1:xxxx:function:custom-errors-in-log-lambda
        [Warning] At least one latest layer version is being used. Ensure to lock in versions for production applications using `--layerVersion` and `--extensionVersion`.

Will apply the following updates:
UpdateFunctionConfiguration -> arn:aws:lambda:us-east-1:xxxx:function:custom-errors-in-log-lambda
{
  "FunctionName": "arn:aws:lambda:us-east-1:xxxx:function:custom-errors-in-log-lambda",
  "Environment": {
    "Variables": {
      "DD_LAMBDA_HANDLER": "lambda_function.lambda_handler",
      "DD_API_KEY": "bf0xxxxeb",
      "DD_SITE": "datadoghq.com",
      "DD_CAPTURE_LAMBDA_PAYLOAD": "false",
      "DD_ENV": "custom-lambda-errors",
      "DD_MERGE_XRAY_TRACES": "false",
      "DD_SERVICE": "custom-lambda-errors",
      "DD_TRACE_ENABLED": "true",
      "DD_VERSION": "0.1",
      "DD_FLUSH_TO_LOG": "true"
    }
  },
  "Layers": [
    "arn:aws:lambda:us-east-1:yyyy:layer:Datadog-Extension:43",
    "arn:aws:lambda:us-east-1:yyyy:layer:Datadog-Python310:73"
  ]
}
TagResource -> arn:aws:lambda:us-east-1:xxxx:function:custom-errors-in-log-lambda
{
  "dd_sls_ci": "v2.14.0"
}
[!] Confirmation needed.
? Do you want to apply the changes? Yes
[!] Instrumenting functions.
√ [us-east-1] Updated 1 Lambda function.

✔ Updated 1 Lambda function.
```

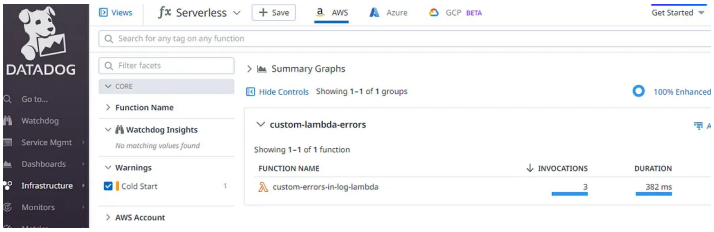
Once installation is completed, we'll see two layers added in our lambda function



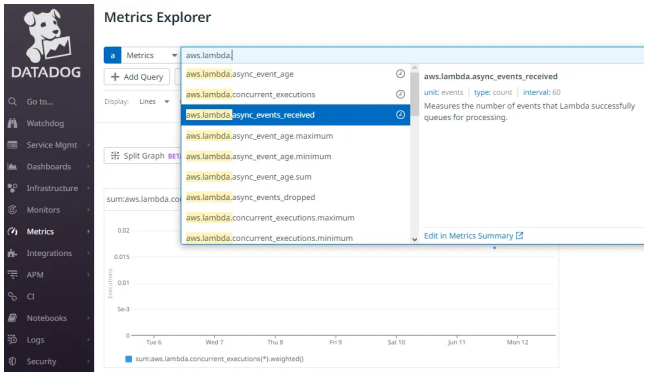
1.2 Configuring Datadog

If you have added your AWS account as integration in Datadog and enabled Lambda integration, you should be able to see this lambda function shows up in Infrastructure -> Serverless page. Otherwise please refer to another post for how to setup AWS integration in Datadog.

Note, keep mind to current time period at top right corner — ONLY lambda functions that have been fired during this time show up.

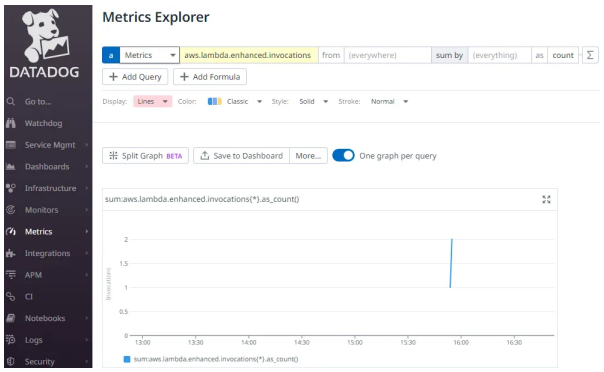


Also on Metric Explorer page, we can see built in metrics are being collected...



Also, we should be able to see “enhanced” lambda metrics with “extension” layers in place like “aws.lambda.enhanced.invocations”. based on Datadog doc.

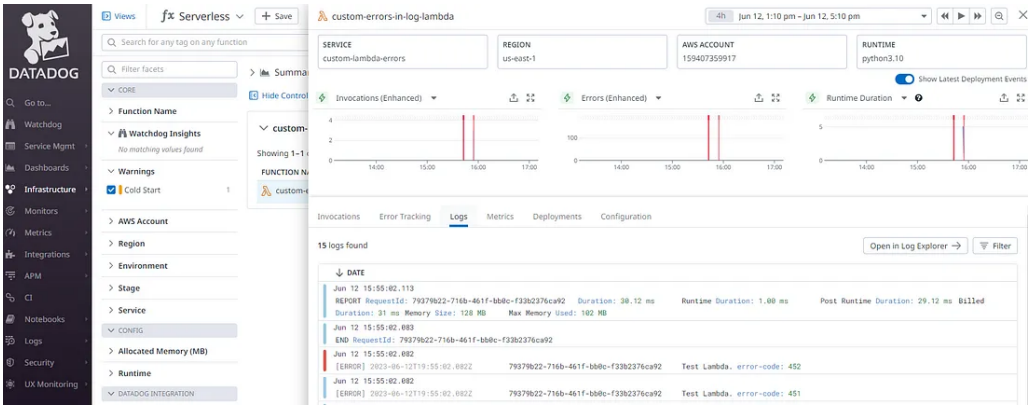
Note, keep mind to current time period at top right corner — the “enhanced” metrics show in the drop down Metrics list ONLY when the lambda functions with “extension” layer have been fired during this time.



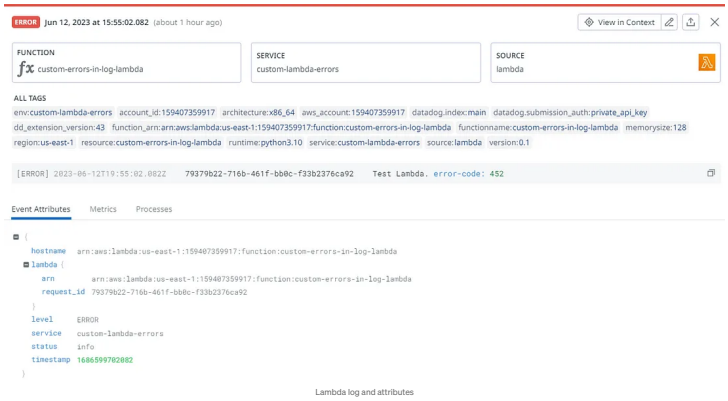
2 Monitor lambda logs for custom error codes

Previously, Datadog uses another Lambda (forwarder Lambda) to forward logs of targeting Lambda from CloudWatch. Now it's recommended to use extension. Extension collects lambda logs directly from Lambda function.

Clicking on the Lambda we just installed extension to, it shows lambda logs in Logs tab.



Clicking the log line with error red color for details



Lambda log and attributes

We see the log line is as follows:

```
[ERROR] 2023-06-12T19:55:02.082Z 79379b22-716b-461f-bb0c-f33b2376ca92 Test Lambda. error-code: 452
```

So this is from our lambda code where we are handling multiple error cases. We want to define our own error codes and save them to log file. For example “error-code: 452”.

Sample Lambda code

```
import json
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    # TODO implement

    #logger.error("Test Lambda. error-code: 451")
    logger.error("Test Lambda. error-code: 452")

    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

The purpose is to have the errors monitored by Datadog. We’d like to count the errors by the custom error code we defined. When count of any kind of errors exceeds a threshold we want an alert in email.

As we know Datadog starts from “Metrics”. On top of “Metrics” we can define “Monitor” which triggers “Alerts”. So how can we create “metric” from custom error code?

Datadog parses log files using built in roles and generate “attribute” from the log line based on it’s understanding. We can see default attributes of Lambda log in above screenshot. They are “hostname”, “level”, “status” etc. “Facet” can be created on top of “attribute” and Metrics can be created on top of “facet”.

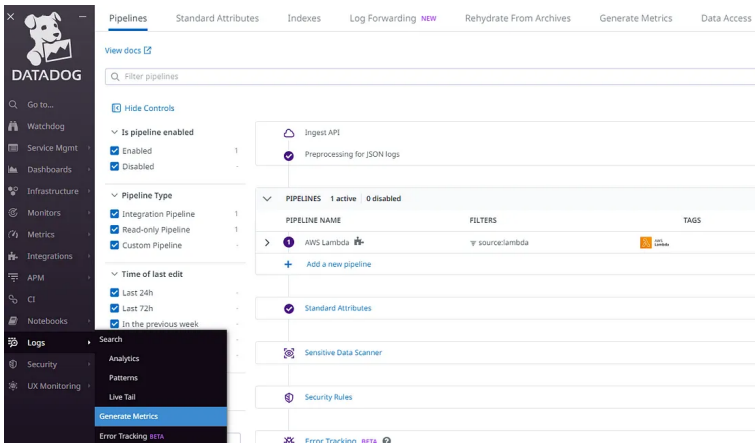
So “log message” -> “attribute” -> “facet” -> “metric” -> (more). Can this be simplified?

But we don’t see our “error-code” as an attribute! So how can we tell Datadog to generate “attributes” for our error code?

We’ll need to create rules based on our custom log format so Datadog can understand us. In Datadog, the configurations are pipeline and processor in the pipeline.

2.1 Create pipeline and add processor for attribute

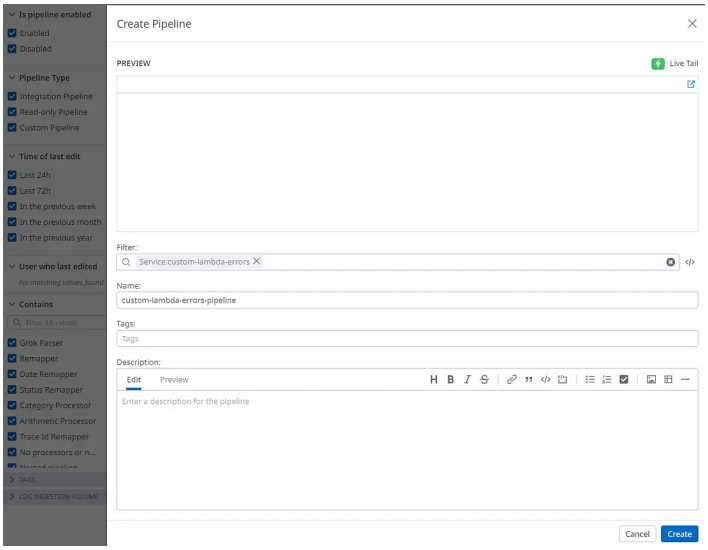
Go to “Logs” -> “Generat Metrics” page and click on “Pipeline” tab.



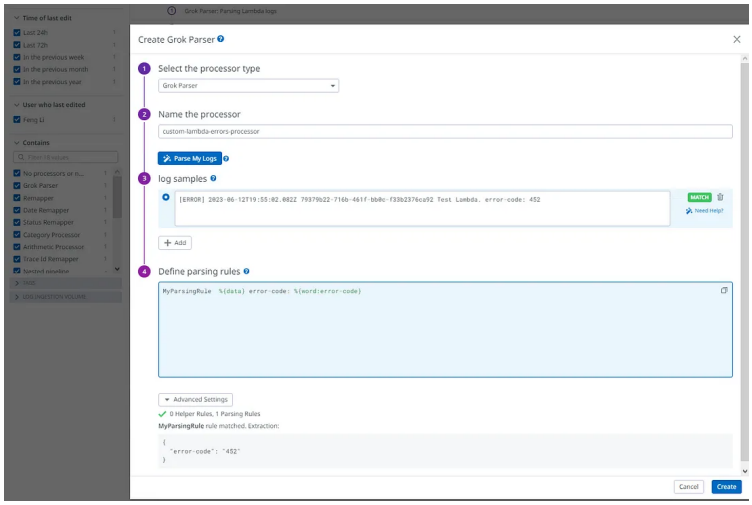
We see there is a built in pipeline called “AWS Lambda” which is used to understand standard Lambda log and find attributes. We’ll create a new pipeline to understand our “error-code” format.

Create a new pipeline called custom-lambda-erros-pipeline. (note, if we don’t have log messages being generating currently we won’t see anything in “Filter” field which is NOT convenient. Datadog uses “Filter” feed pipeline given logs.

I’ll key in our service name “custom-lambda-errors”. This “service” name can be used by Datadog to categorize logs. We have set this “service” name when installing “extension” to our lambda. Now with this filter set, only our logs go through this new pipeline.



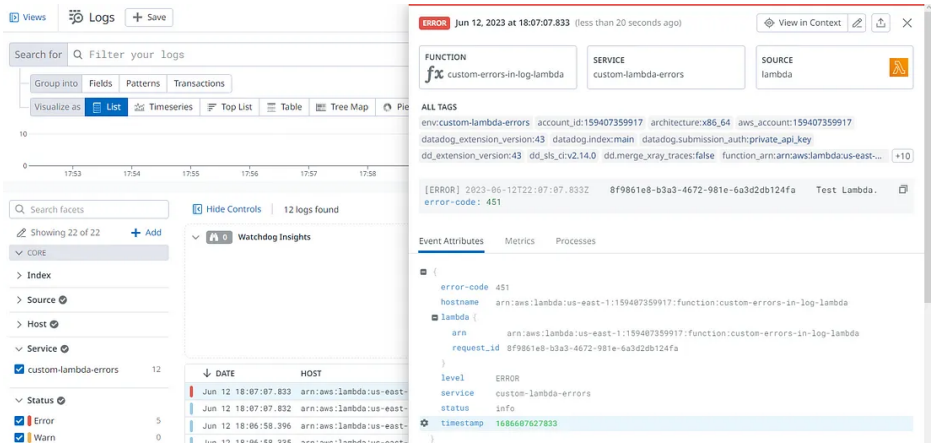
Now click the pipeline name and use “Add Processor” link underneath to create a processor. Choose “Grok Parser” as type, give a name “custom-lambda-erros” and past one line of our log message.



With parsing rule as following we can see “error-code” is treated as an attribute. We used “word” to extract error code as it’s qualitative value. In case of quantitative value we need to use “number”.

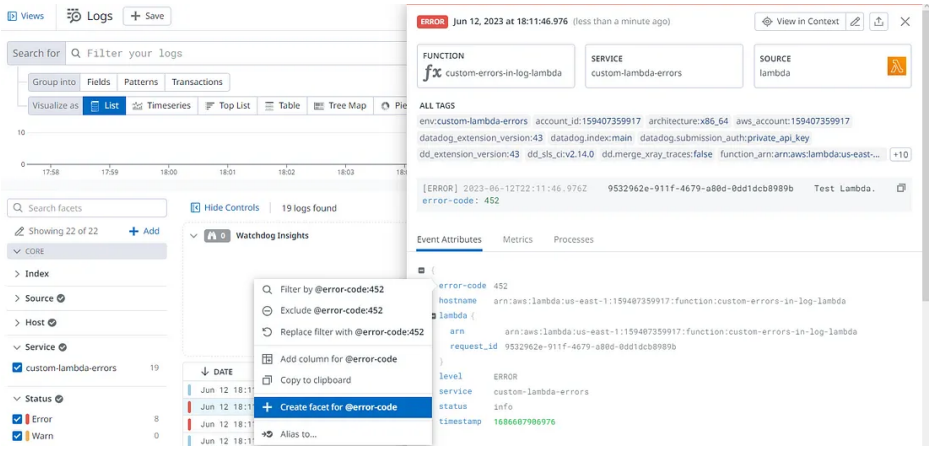
```
MyParsingRule  %{data} error-code: %{word:error-code}
```

Now fire the lambda to generate new logs…now we see “error-code” is a new added attribute.

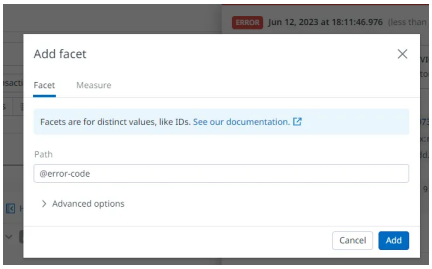


2.2 Create Facet based on attribute

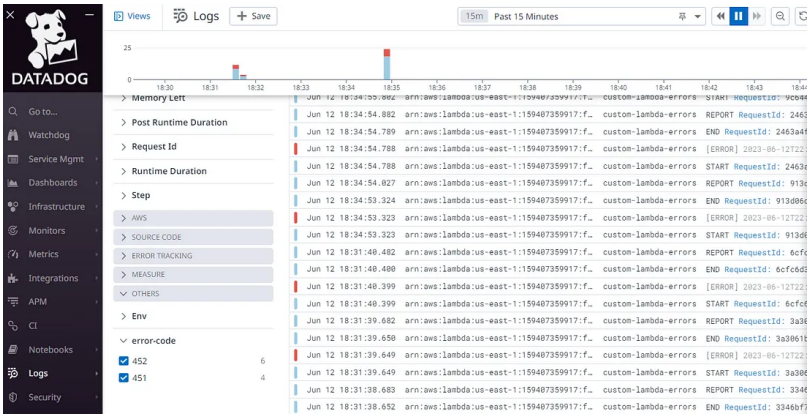
Right click the attribute “error-code” to create “facet”.



Use default name “@error-code” for the facet.

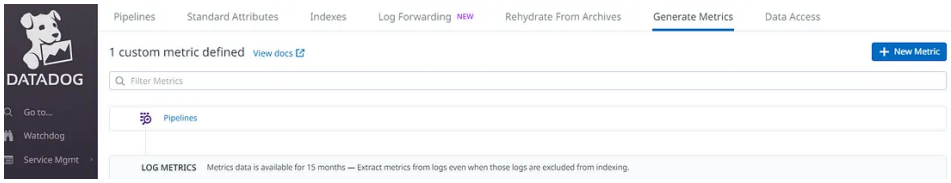


With this facet we can filter logs on log page...

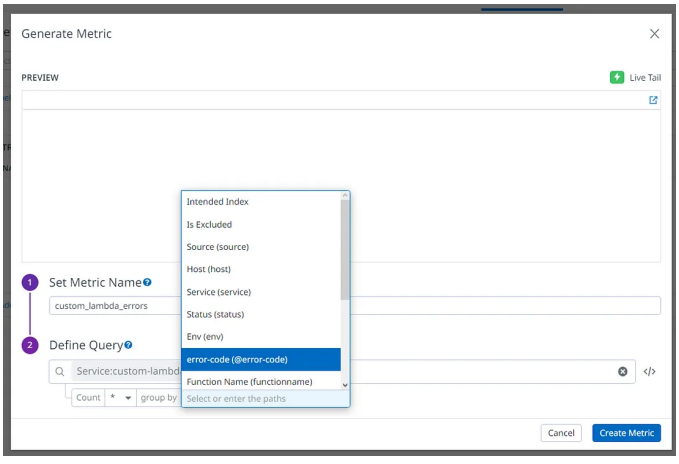


2.3 Create Metric based on facet

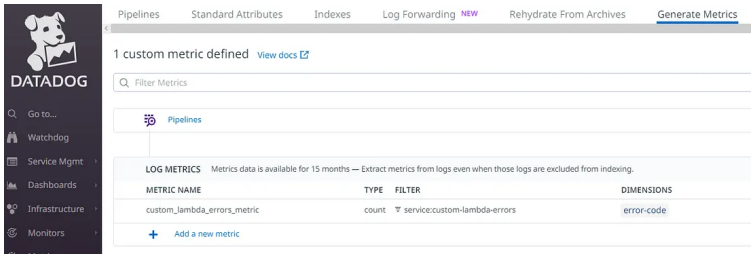
Go to Logs -> Generate Metrics page. Click on “New Metric” button.



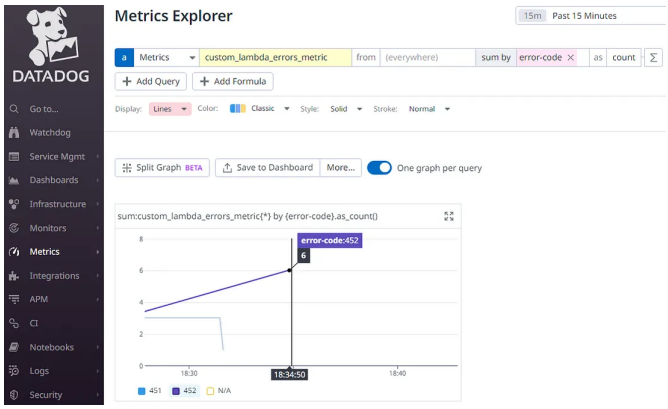
Provide name “custom_lambda_errors_metric” and choose above facet.



Now we have create a new metric for our custom error code



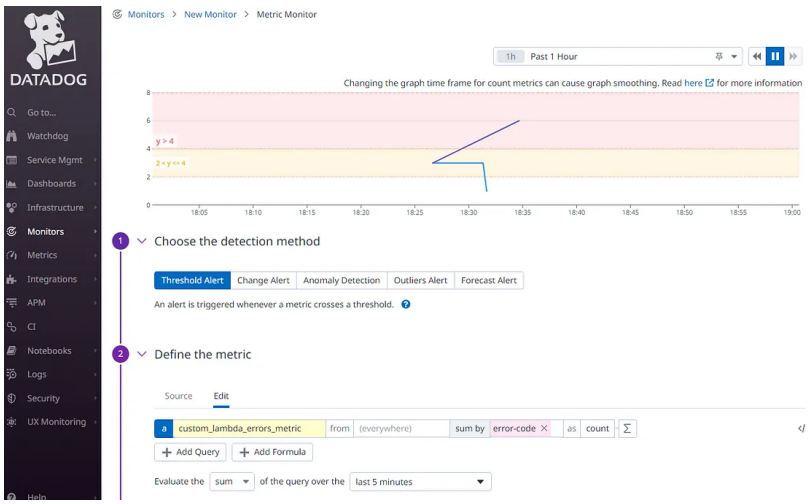
Take a look at our custom metric in metrics explorer after firing lambda some more ...

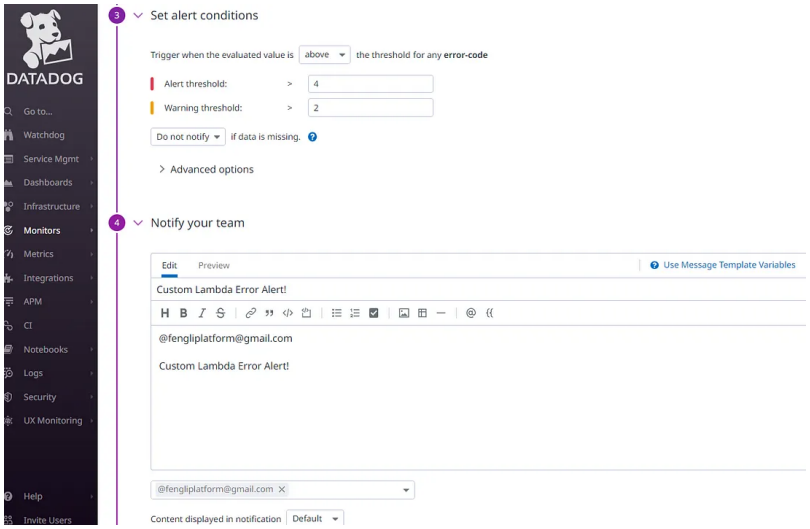


2.4 Create monitor/alert based on the metric

We'd like to create a monitor for count of each error code. When counts exceed threshold we want to receive email notifications.

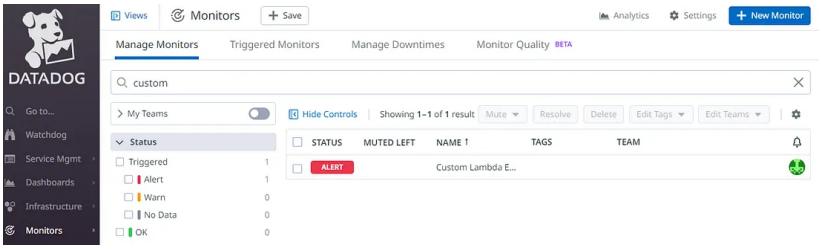
Go to "Monitors" -> "New monitor" page, choose "Threshold Alert" and our custom metric. Choose "error-code" facet for sum by.



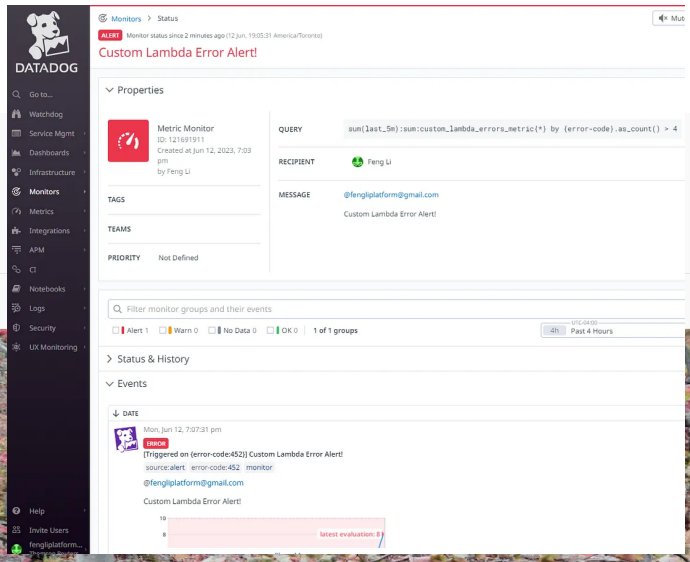


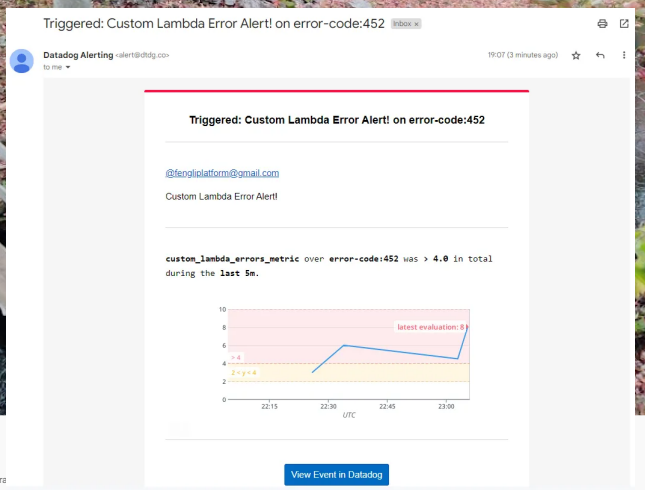
There are more options when defining monitor. we use default for most of the rest options.

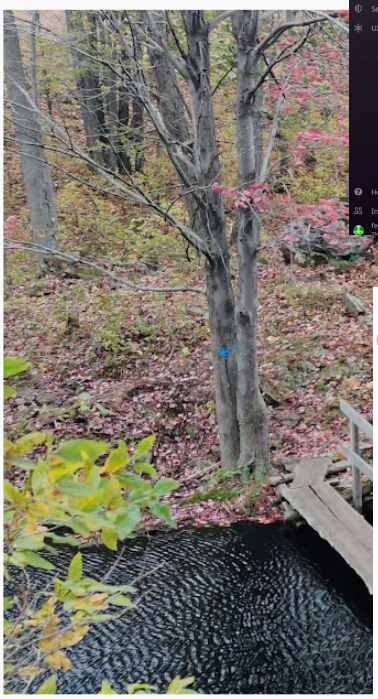
After creating monitor, we fire some more errors in our lambda. Datadog shows red alert now!

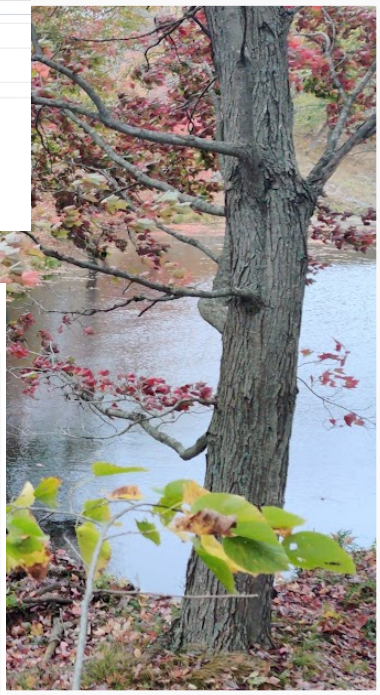


Datadog AWS Lambda Log Monitoring









Follow

Written by Feng Li

314 Followers

Big Data Engineer at Thomson Reuters, Jogger, Hiker. https://www.linkedin.com/in/fli01/

More from Feng Li

Feng Li

DBT project with Snowflake

DBT acts as T in ETL/ELT. DBT framework includes SQL and Jinja. SQL is core of data tra

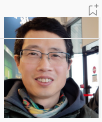
8 min read · 6 days ago

Happy Reading!

Join Medium with my referral link - Feng Li

It started with personal study notes with no pressure of perfection... After a while folks said some posts are somehow...

medium.com



Feng Li

Play with Snowflake Direct Share

Direct share has interesting features. Here are some brief discussions.

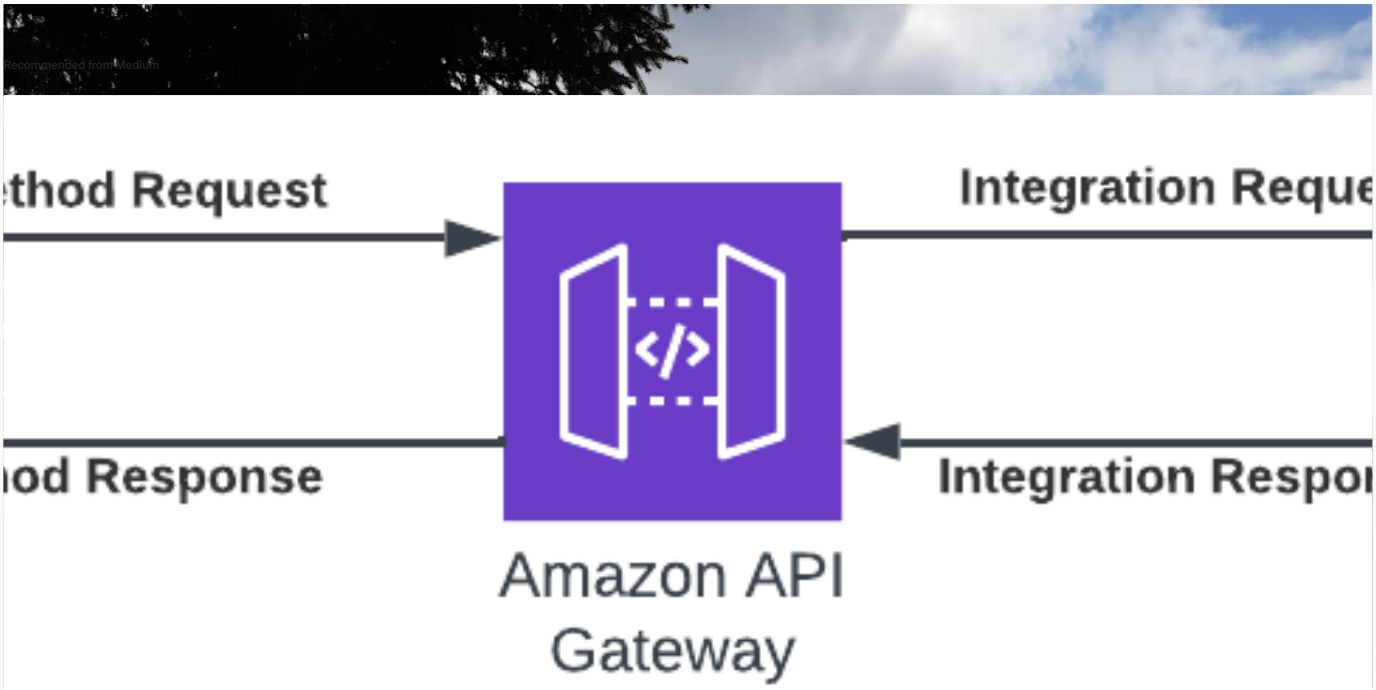
· 5 min read · Sep 29

Feng Li in Dev Genius

Cursor and for loop in a Snowflake stored procedure

Snowflake scripting can be tricky in terms of the syntax. In sql language based stored procedure there is some learning curve.

· 3 min read · Feb 28



Simplecloudquestions.com in AWS Tip

Tutorial: Building an AWS API Gateway with Lambda Function

Introduction:

4 min read · Jun 19

Rushabh Trivedi

Integrating Chat-GPT with Lambda

Its been while I was reading about ChatGPT and its usefulness over the tech blogs. I was curious on how to integrate ChatGPT with the...

4 min read · Jun 9

Winnie Ho in AMA Technology Blog

Structured Logging using AWS Lambda Powertools for TypeScript

Understand application state and events.

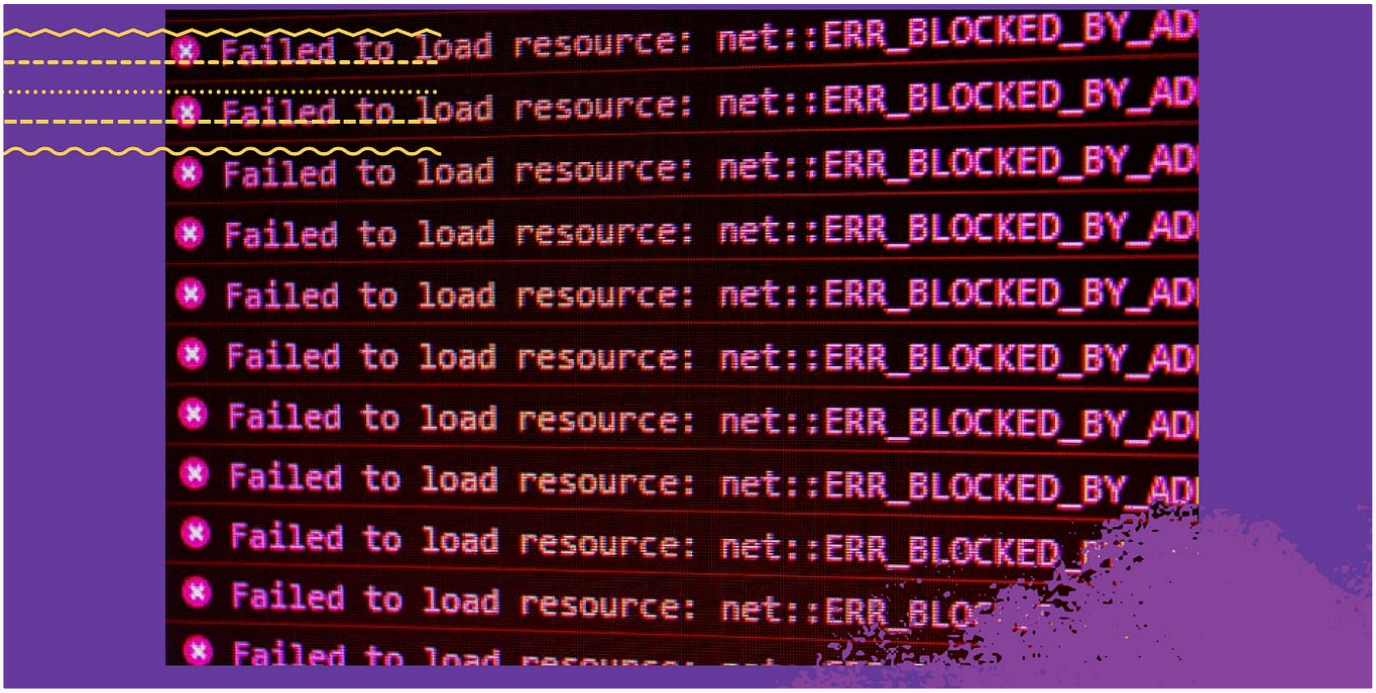
6 min read · May 15

Louis G in Towards Data Engineering

Creating AWS Lambda Layers for Python

How to create and deploy a Lambda Layer using CloudFormation to promote modularity, isolation and simplicity in your Python code.

7 min read · Jun 3

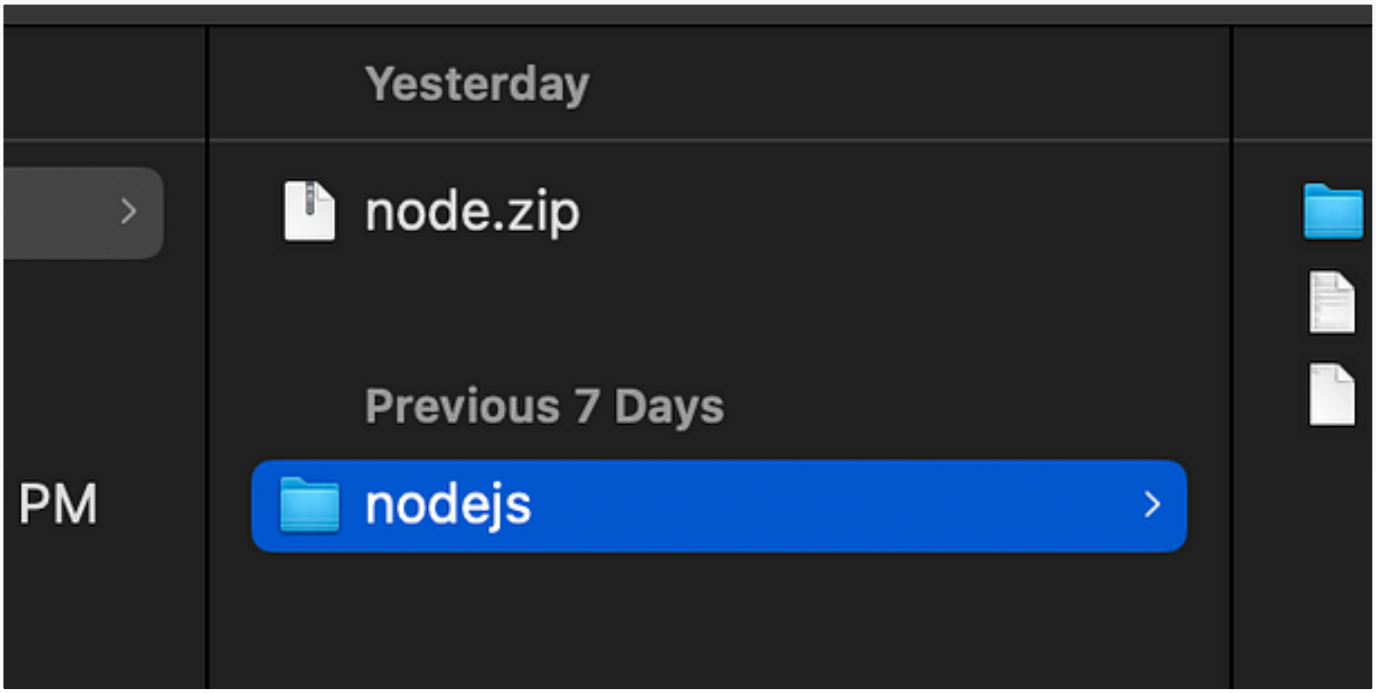


Frederick Swadling in Creating TotallyMoney

Error handling in AWS Lambda

Ensuring robust error reporting in large scale systems built on AWS Lambda

12 min read · May 24



Anuragchitti

Creating Lambda Layer with Node.js

Creating Lambda Layer with Node.js

2 min read · Jul 18

See more recommendations