

AWS Cloud Operations & Migrations Blog

Introducing CloudWatch Lambda Insights

by Amit Kalawat | on 03 NOV 2020 | in [Advanced \(300\)](#), [Amazon CloudWatch](#), [Announcements](#), [Learning Levels](#), [Management Tools](#) | [Permalink](#) | [Share](#)

CloudWatch Lambda Insights is a monitoring and troubleshooting solution for serverless applications running on AWS Lambda. The solution collects, aggregates, and summarizes system-level metrics including CPU time, memory, disk, and network. It also collects, aggregates, and summarizes diagnostic information such as cold starts and Lambda worker shutdowns to help you isolate issues with your Lambda functions and resolve them quickly.

Lambda Insights uses a new CloudWatch Lambda extension, which is provided as a Lambda layer. When you install this extension on a Lambda function, it collects system-level metrics and emits a single performance log event to CloudWatch Logs for every invocation of that Lambda function. The performance events leverage the embedded metrics format to collocate metrics and metadata. For more information about Lambda extensions, see [Using AWS Lambda extensions](#). For more information about embedded metric format, see [Ingesting High-Cardinality Logs and Generating Metrics with CloudWatch Embedded Metric Format](#).

Enabling Lambda Insights

You can use the Lambda Console to enable Lambda Insights on a given function. Alternatively, you can use the AWS CLI, AWS CloudFormation, the AWS Serverless Application Model CLI (AWS SAM), or the AWS Cloud Development Kit (AWS CDK). Please check the documentation

at <https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/Lambda-Insights-Getting-Started.html>.

Let's get started

In this blog post, we will create a few "Hello World" Lambda functions and monitor them using Lambda Insights. We will be using the AWS CDK to deploy our architecture.

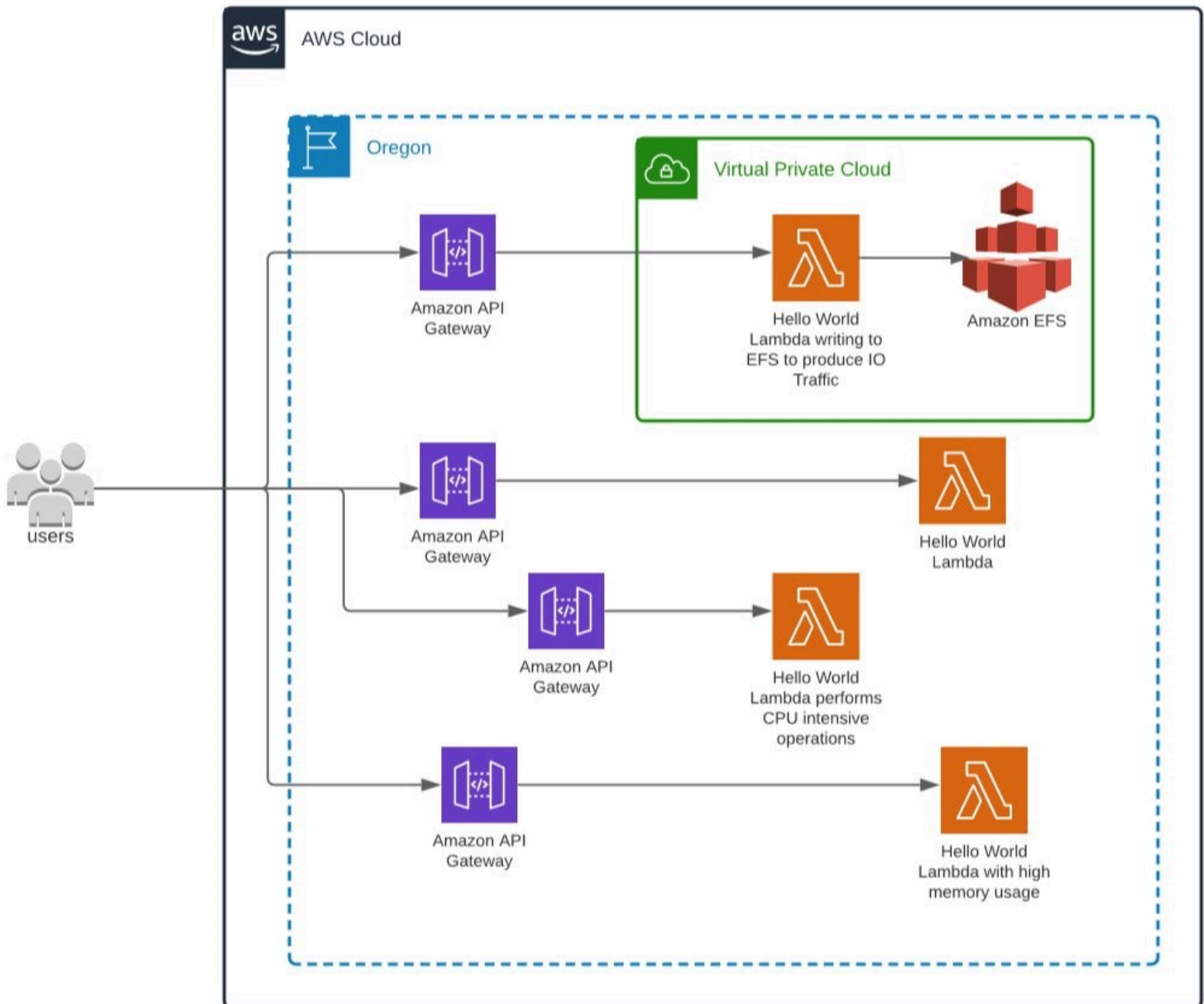
Deploy the architecture

Run the following commands to clone the Git repository and start the deployment. In this example, I am using Cloud9 (<https://aws.amazon.com/cloud9/>) to perform the deployment. You are welcome to use your favorite terminal.

```
Bash
git clone https://github.com/aws-samples/aws-cdk-quickstart-lambda-insights
cd aws-cdk-quickstart-lambda-insights
npm install
cd resources && npm install
cdk bootstrap
cdk deploy
```

The AWS CDK will deploy the architecture below. It will include:

- A Lambda function which outputs “Hello World!” each time it’s invoked.
- A Lambda function which runs in a VPC and writes dummy data to an EFS volume to mimic network traffic. The Lambda function outputs “Hello World – EFS Lambda!” each time it’s invoked.
- A Lambda function which performs CPU-intensive operations. We are calculating Fibonacci sequences during the runtime. The Lambda function outputs “Hello World – CPU!” each time it’s invoked.
- A Lambda function which performs memory-intensive operations. We will do that by loading arrays during function runtime. The Lambda function outputs “Hello World – Memory!” each time it’s invoked.
- Each of the Lambda functions will be invoked by an Amazon API Gateway.



Generating Traffic / Test Data

On successful completion of the AWS CDK deployment, you will see a message like this:

```

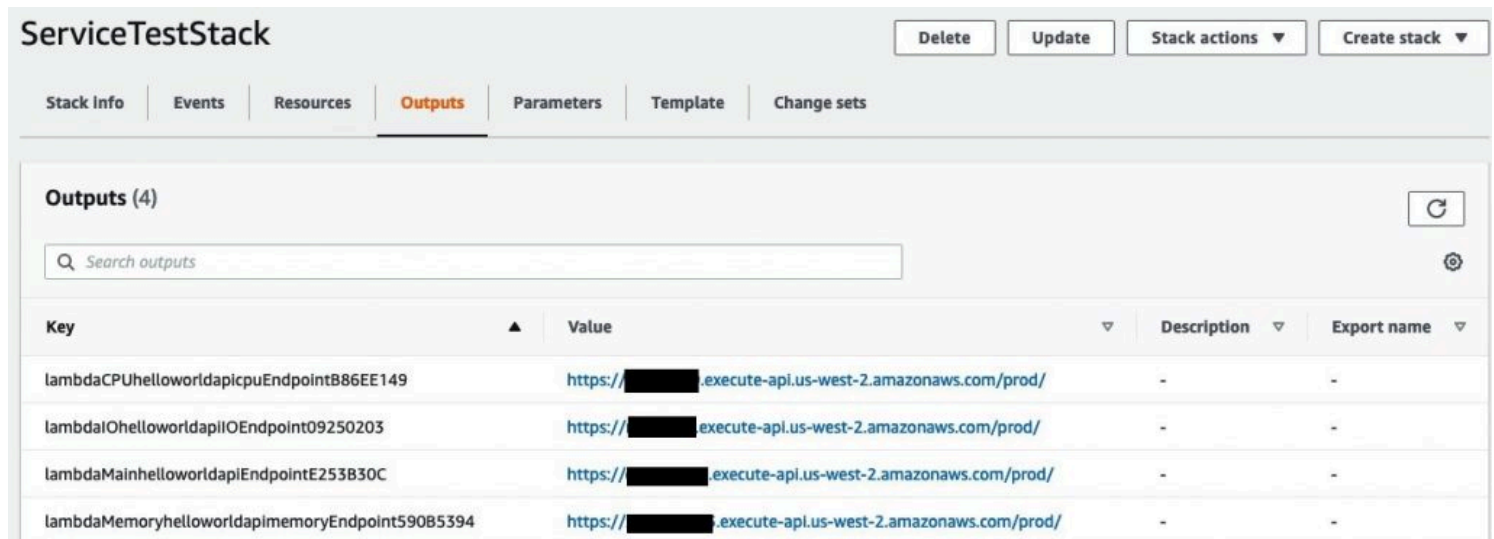
ServiceTestStack

Outputs:
ServiceTestStack.lambdaCPUhelloworldapicpuEndpointB86EE149 = https://[redacted].execute-api.us-west-2.amazonaws.com/prod/
ServiceTestStack.lambdaIOhelloworldapiIOEndpoint09250203 = https://[redacted].execute-api.us-west-2.amazonaws.com/prod/
ServiceTestStack.lambdaMainhelloworldapiEndpointE253B30C = https://[redacted].execute-api.us-west-2.amazonaws.com/prod/
ServiceTestStack.lambdaMemoryhelloworldapimemoryEndpoint590B5394 = https://[redacted].execute-api.us-west-2.amazonaws.com/prod/

Stack ARN:
arn:aws:cloudformation:us-west-2:[redacted]:stack/ServiceTestStack/[redacted]

```

Take a note of each of the 4 URLs. Alternatively, you can also go to the CloudFormation console and search for ServiceTestStack and check outputs. We will use these 4 URLs to generate traffic for each Lambda function using CloudWatch Synthetics.

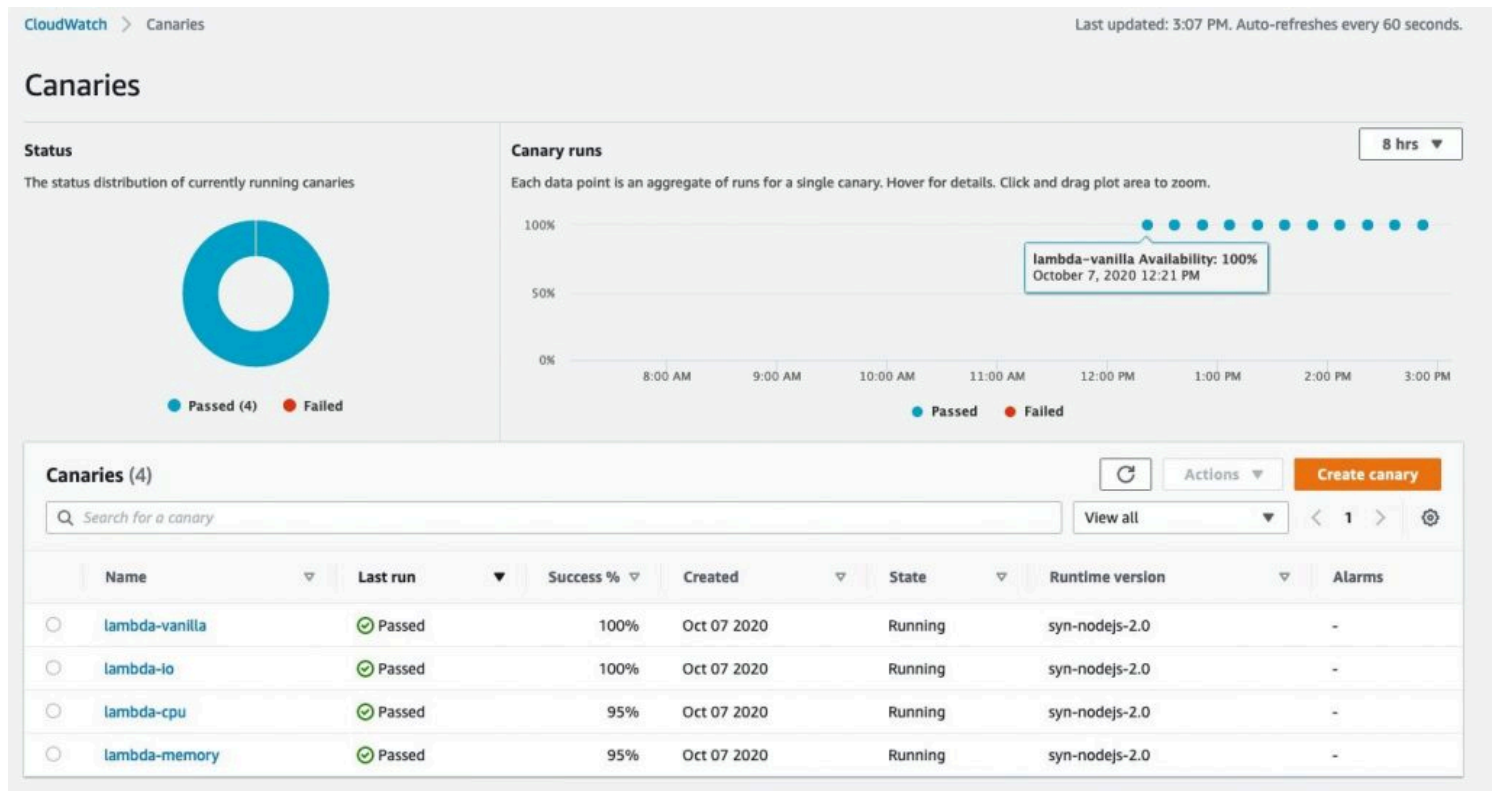


| Key | Value | Description | Export name |
|---|--|-------------|-------------|
| lambdaCPUhelloworldapicpuEndpointB86EE149 | https://[redacted].execute-api.us-west-2.amazonaws.com/prod/ | - | - |
| lambdaIOhelloworldapiIOEndpoint09250203 | https://[redacted].execute-api.us-west-2.amazonaws.com/prod/ | - | - |
| lambdaMainhelloworldapiEndpointE253B30C | https://[redacted].execute-api.us-west-2.amazonaws.com/prod/ | - | - |
| lambdaMemoryhelloworldapimemoryEndpoint590B5394 | https://[redacted].execute-api.us-west-2.amazonaws.com/prod/ | - | - |

For each of the 4 URLs, do the following:

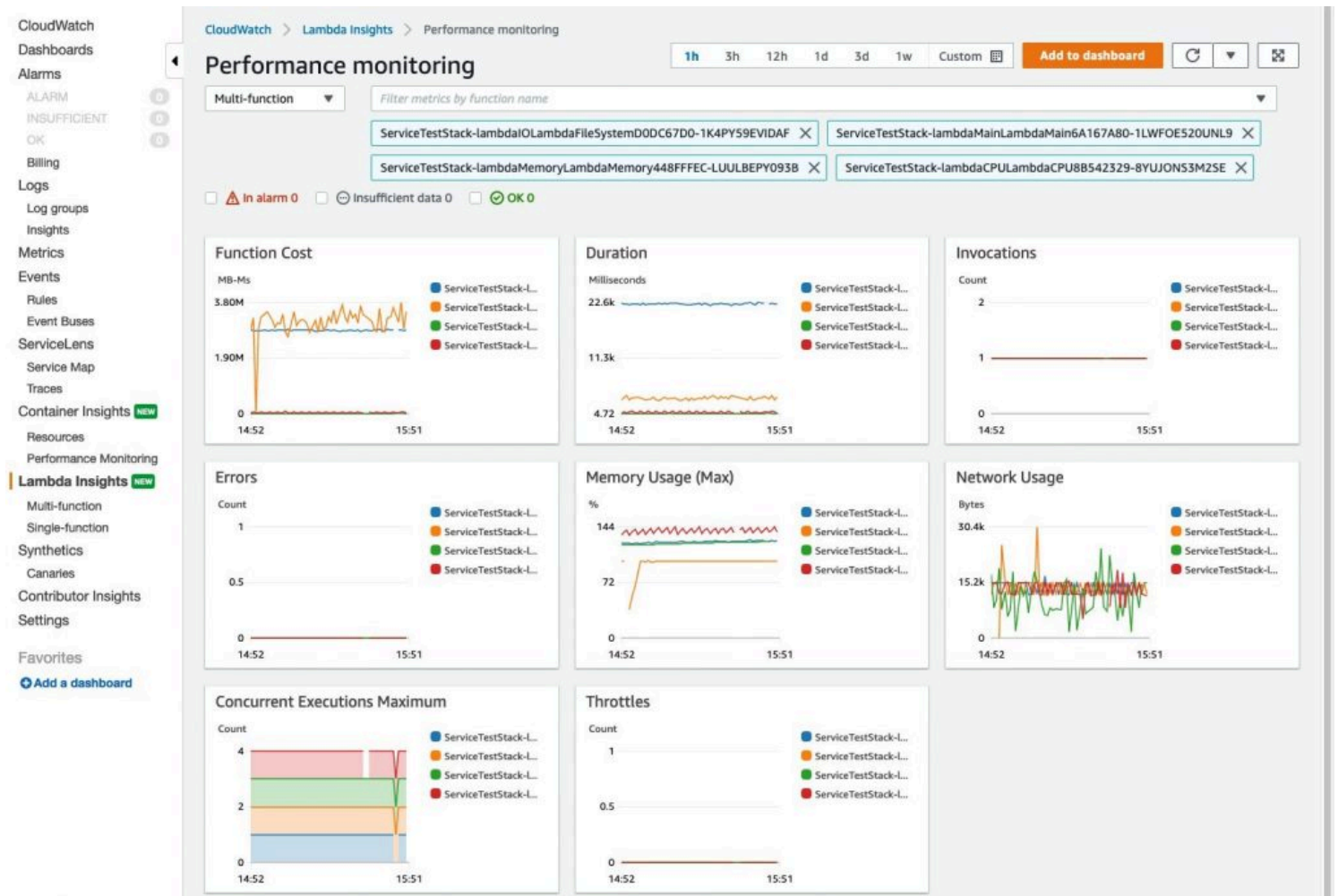
1. Go to CloudWatch → Synthetics → Canaries and select "Create Canary."
2. Select the default options of "Use a Blueprint" and "Heartbeat Monitoring."
3. Enter any name under Name and paste one of the URLs from CloudFormation/CDK outputs in "Application or Endpoint URL."
4. Keep scrolling down and under "
5. Keep the rest of the options as default and select "Create canary" at the bottom of the screen. You will have to wait for 1 minute before the canary is created and starts monitoring your API.

Once you have created canaries for each of the 4 URLs, your CloudWatch → Canaries screen should look like:

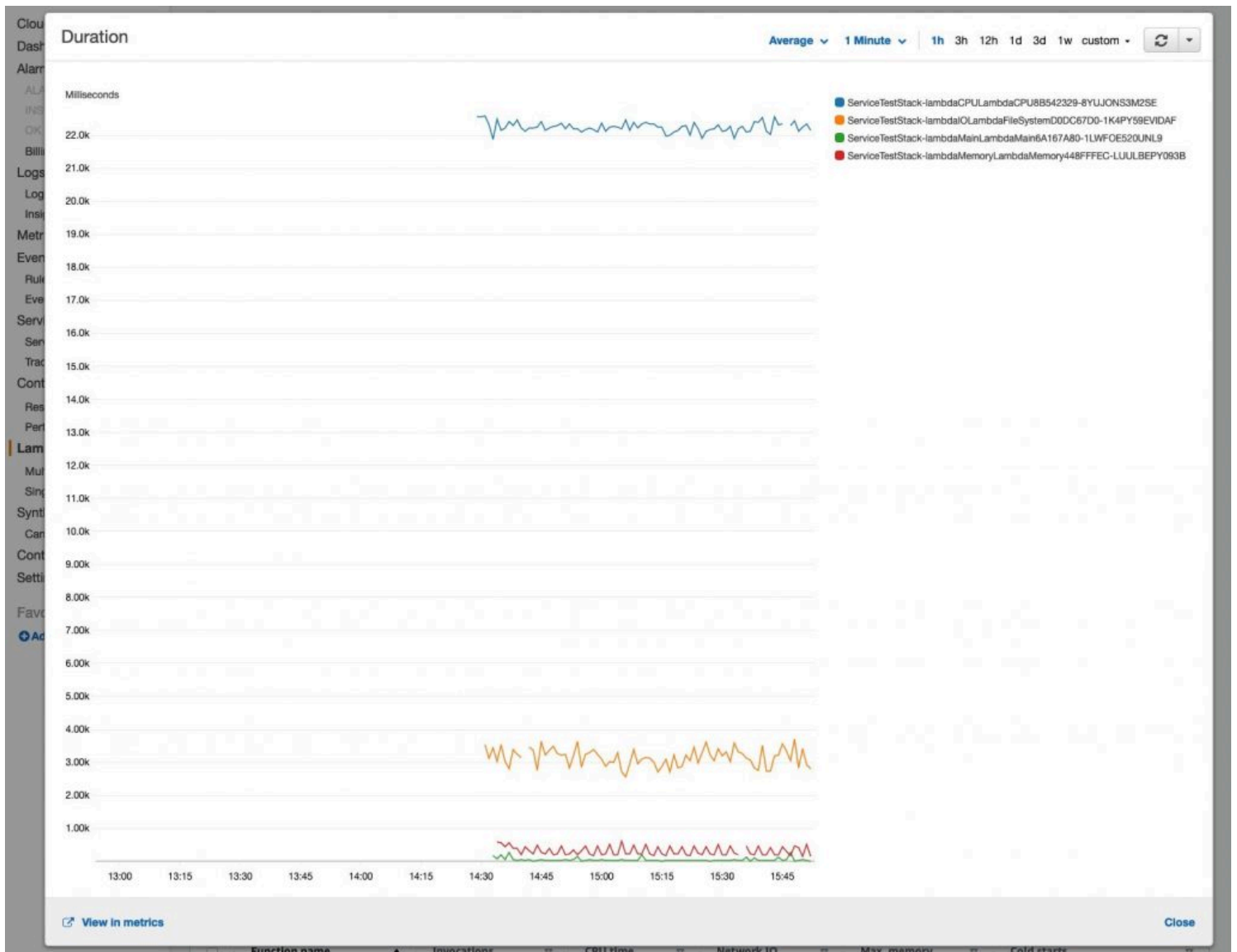


Using Lambda Insights (Multi-Function)

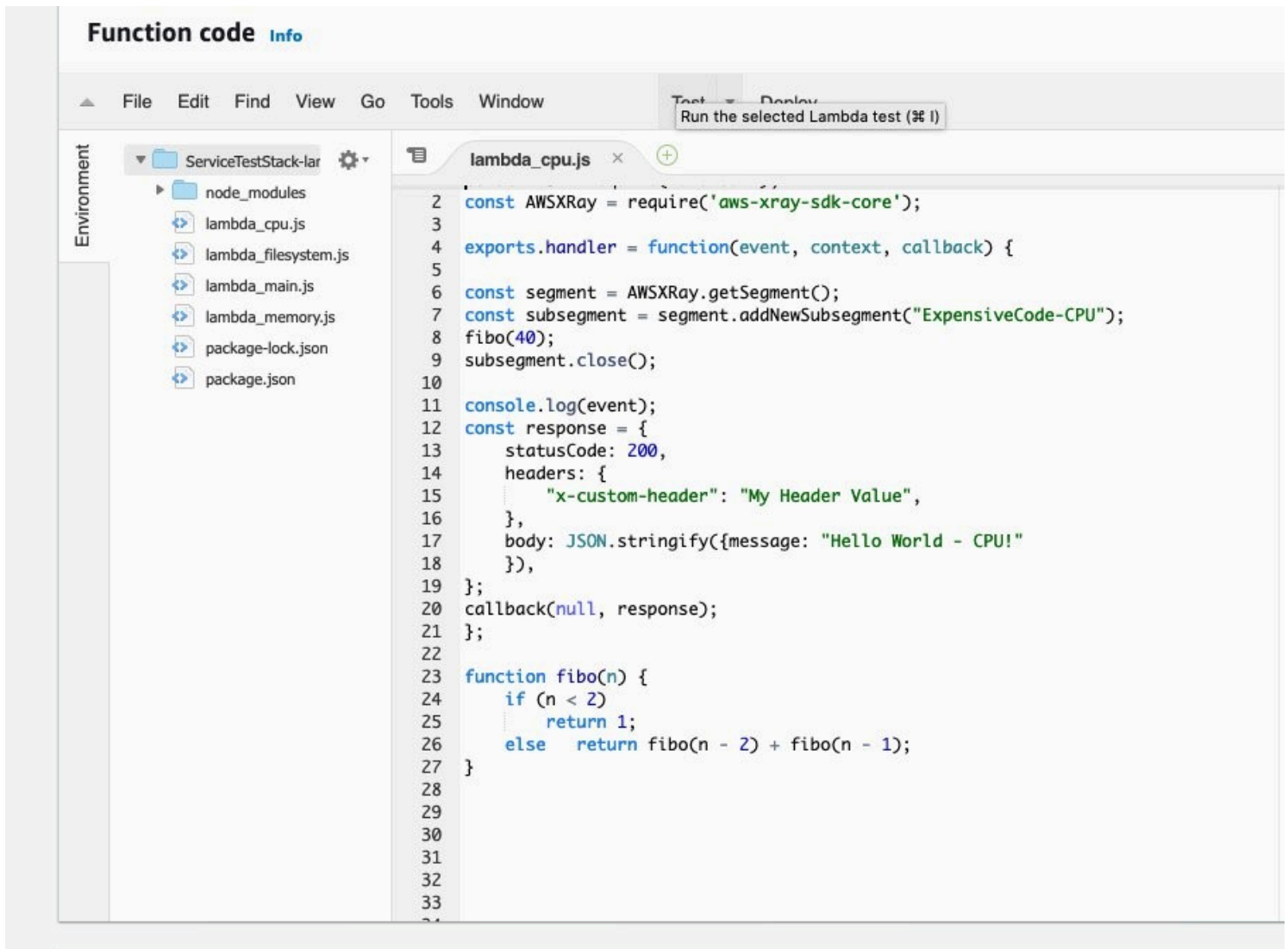
We can go ahead and start looking at Lambda Insights in CloudWatch. To do this, go to CloudWatch and click on “Multi-function” under Lambda Insights. Here, we can observe metrics for the four Lambda functions created by the AWS CDK.



Double-clicking on Duration, we see that the Lambda function "ServiceTestStack-lambdaCPU" is consuming more time.



The Duration metric is comparatively higher than the other due to CPU-intensive statements. Check line number 8 for the Fibonacci function.



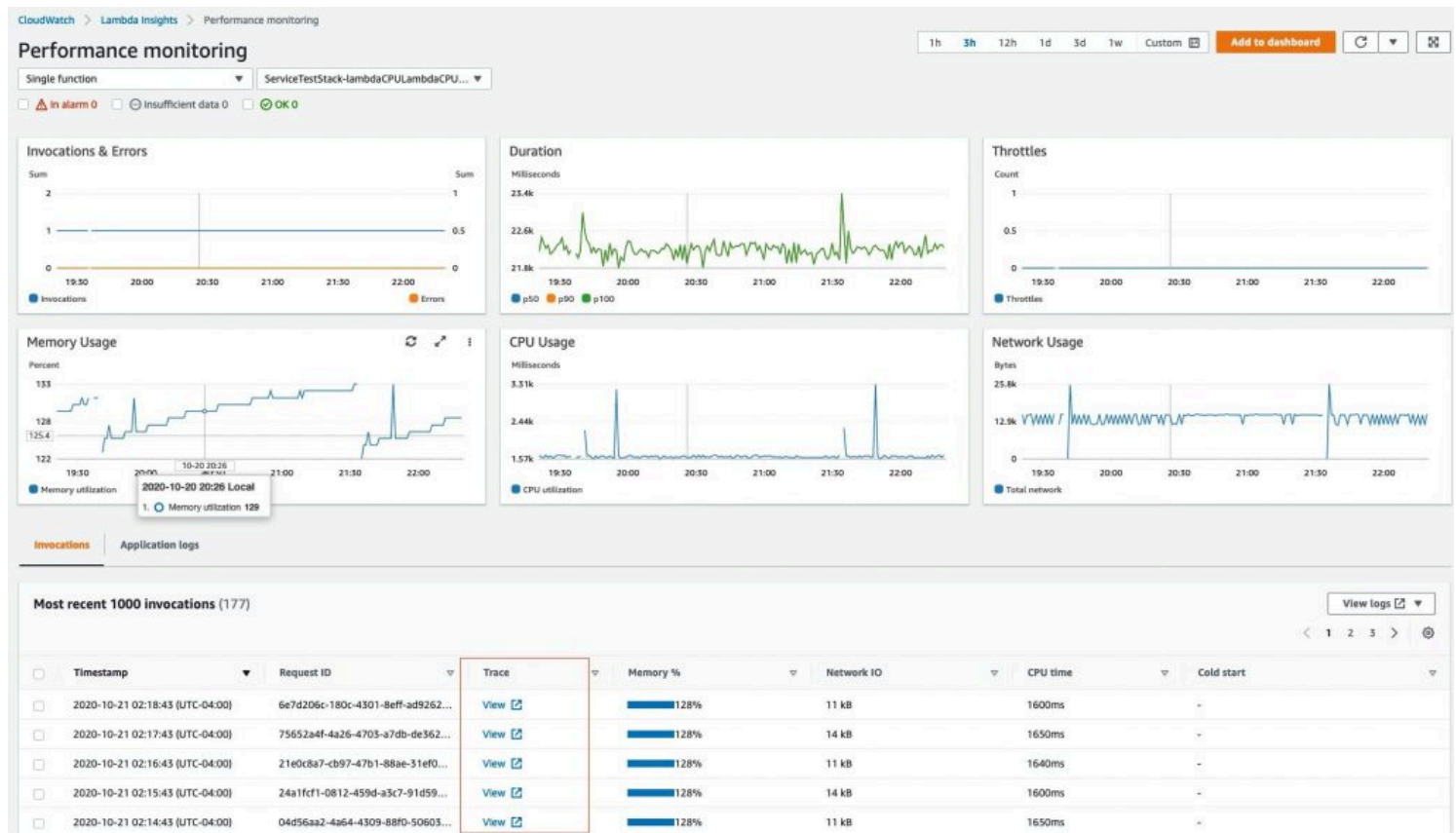
Using Lambda Insights (Single Function)

At the bottom of the Lambda Insights Multi-Function page, you can select any of the given Lambda functions to see metrics on a per function basis. Let's select the Lambda which has the phrase "CPULambda" in it:

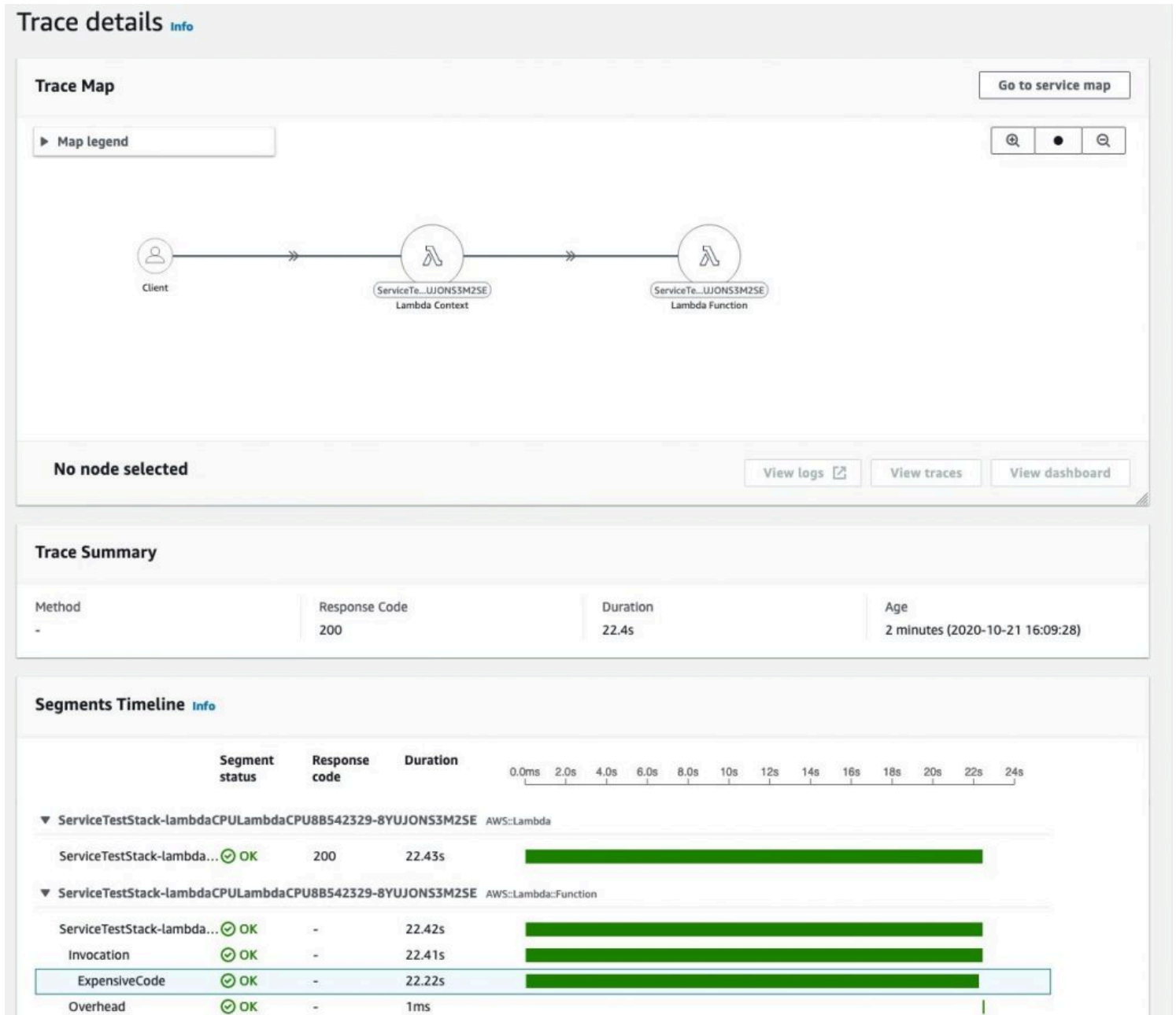
Function summary (4)

| <input type="checkbox"/> | Function name | Invocations | CPU time | Network IO |
|--------------------------|--|-------------|-------------|------------|
| <input type="checkbox"/> | ServiceTestStack-lambdaCPULambdaCPU8B542329-801D2O0NRTL | 178 | 1643.9326ms | 2 MB |
| <input type="checkbox"/> | ServiceTestStack-lambdaMainLambdaMain6A167A80-1JK9C5LTOO1OH | 176 | 950.625ms | 2 MB |
| <input type="checkbox"/> | ServiceTestStack-lambdaMemoryLambdaMemory448FFEC-1JCCDGBIM632A | 177 | 141.4124ms | 1 MB |
| <input type="checkbox"/> | ServiceTestStack-lambdaFilesystemLambdaFilesystemD0DC67D0-VQLDBEOKSK2F | 178 | 146.2921ms | 2 MB |

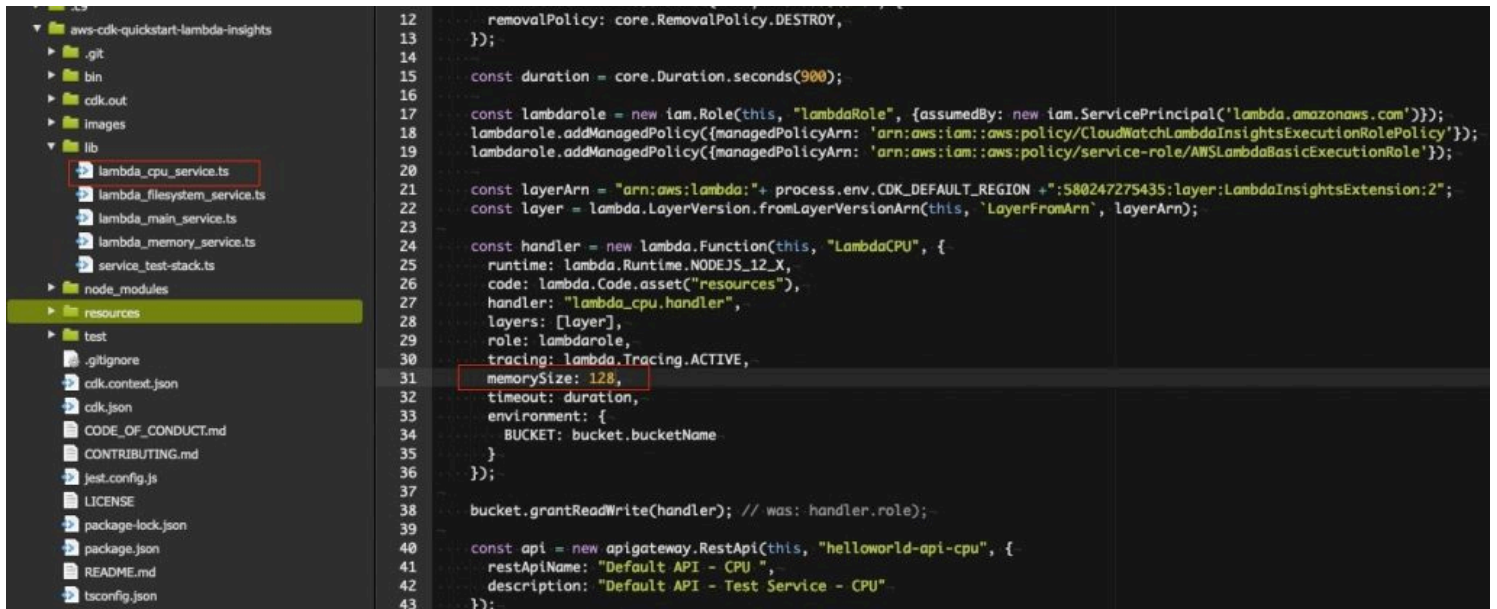
On the next screen, we can see CPU, memory, and network utilization for the selected Lambda function. If you have enabled AWS X-Ray for your Lambda function, the trace for each Lambda execution will be available in the Trace column.



On this page, we are seeing that the function is fully consuming the memory allocated to the Lambda runtime. Viewing a trace also reveals that a significant portion of time is spent in an expensive AWS X-Ray segment (almost 22 seconds).



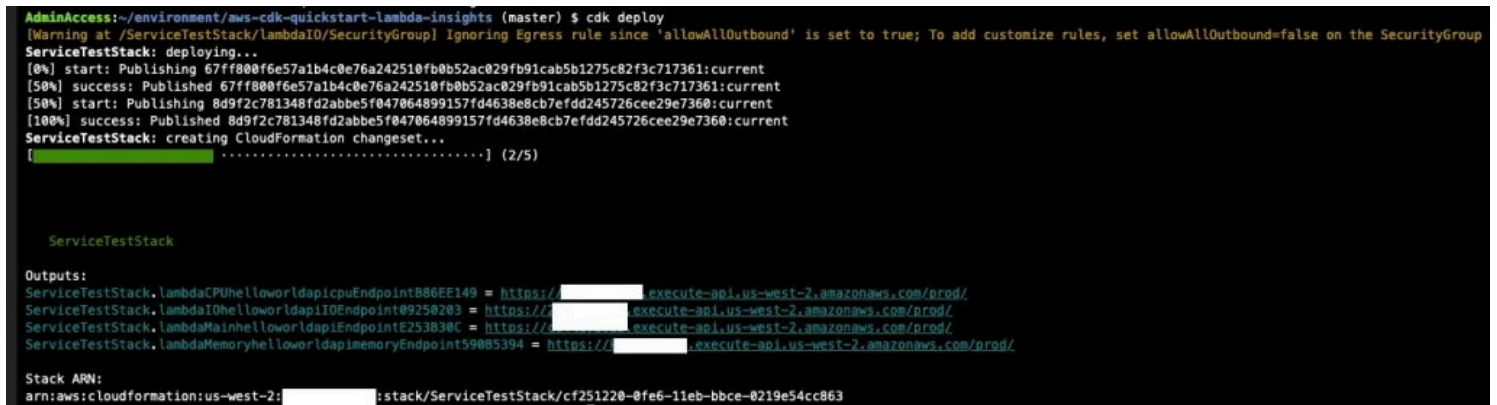
We will increase the memory allocated to this Lambda function to 512 MB. We can do this in the AWS CDK by going to the folder where the Git repository is cloned and opening the `lambda_cpu_service.ts` file under `Lib`. Change the value of 128 to 512 and save the file. You will need to do “`cdk deploy`” to push the changes to your Lambda function.



```

12     removalPolicy: core RemovalPolicy.DESTROY,
13   });
14
15   const duration = core Duration.seconds(900);
16
17   const lambdaRole = new iam.Role(this, "lambdaRole", { assumedBy: new iam.ServicePrincipal('lambda.amazonaws.com') });
18   lambdaRole.addManagedPolicy({ managedPolicyArn: 'arn:aws:iam::aws:policy/CloudWatchLambdaInsightsExecutionRolePolicy' });
19   lambdaRole.addManagedPolicy({ managedPolicyArn: 'arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole' });
20
21   const layerArn = `arn:aws:lambda:${process.env.CDK_DEFAULT_REGION}:${S80247275435:layer:LambdaInsightsExtension:2}`;
22   const layer = lambda.LayerVersion.fromLayerVersionArn(this, 'LayerFromArn', layerArn);
23
24   const handler = new lambda.Function(this, "LambdaCPU", {
25     runtime: lambda.Runtime.NODEJS_12_X,
26     code: lambda.Code.asset("resources"),
27     handler: "lambda_cpu.handler",
28     layers: [layer],
29     role: lambdaRole,
30     tracing: lambda.Tracing.ACTIVE,
31     memorySize: 128,
32     timeout: duration,
33     environment: {
34       BUCKET: bucket.bucketName
35     }
36   });
37
38   bucket.grantReadWrite(handler); // was: handler.role);
39
40   const api = new apigateway.RestApi(this, "helloworld-api-cpu", {
41     restApiName: "Default API - CPU ",
42     description: "Default API - Test Service - CPU"
43   });

```



```

AdminAccess:~/environment/aws-cdk-quickstart-lambda-insights (master) $ cdk deploy
[Warning at /ServiceTestStack/lambdaIO/SecurityGroup] Ignoring Egress rule since 'allowAllOutbound' is set to true; To add customize rules, set allowAllOutbound=false on the SecurityGroup
ServiceTestStack: deploying...
[0%] start: Publishing 67ff800f6e57a1b4c0e76a242510fb0b52ac029fb91cab5b1275c82f3c717361:current
[50%] success: Published 67ff800f6e57a1b4c0e76a242510fb0b52ac029fb91cab5b1275c82f3c717361:current
[50%] start: Publishing 8d9f2c781348fd2abbe5f047064899157fd4638e8cb7efdd245726cee29e7360:current
[100%] success: Published 8d9f2c781348fd2abbe5f047064899157fd4638e8cb7efdd245726cee29e7360:current
ServiceTestStack: creating CloudFormation changeset...
[.....] (2/5)

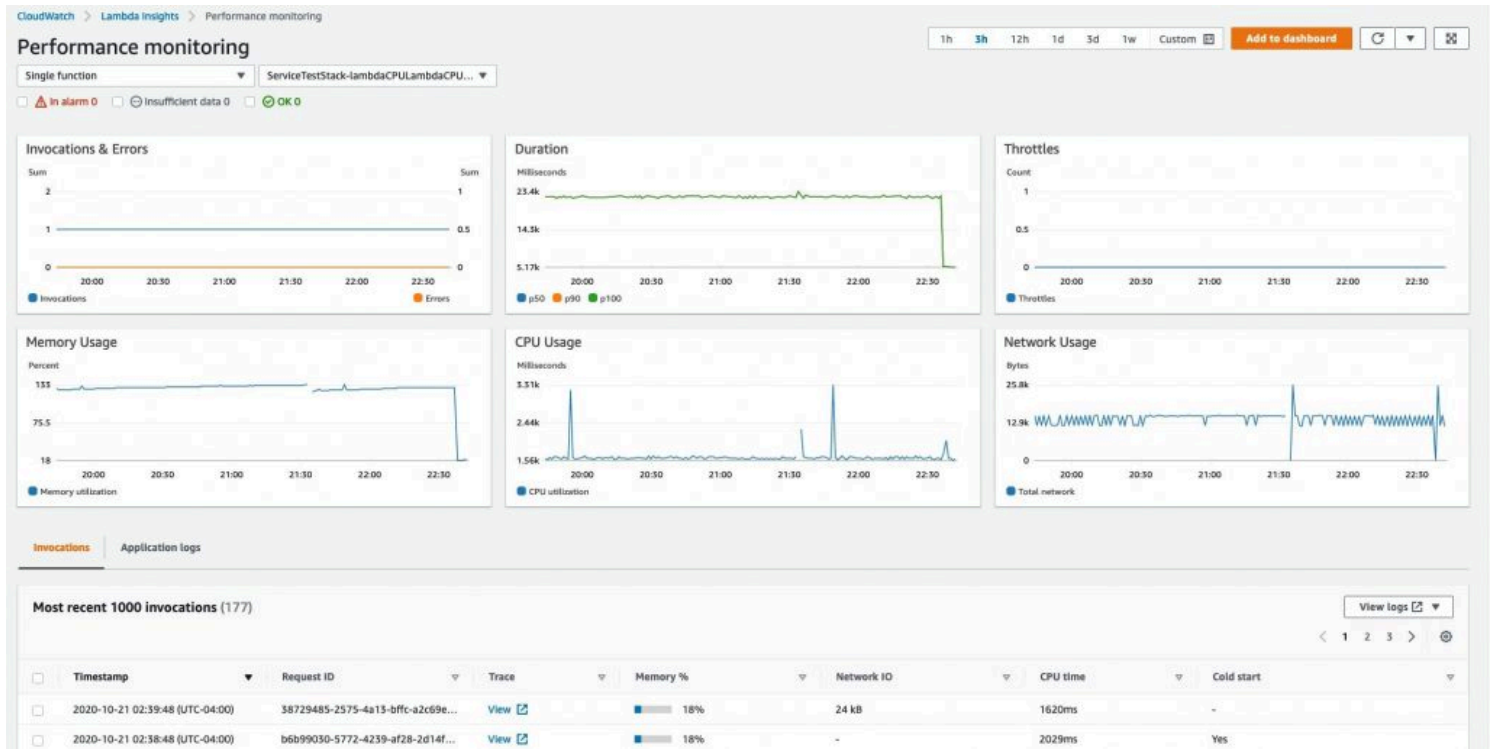
ServiceTestStack

Outputs:
ServiceTestStack.lambdaCPUhelloworldapicpuEndpoint886EE149 = https://[redacted].execute-api.us-west-2.amazonaws.com/prod/
ServiceTestStack.lambdaIOhelloworldapiIIOEndpoint09250203 = https://[redacted].execute-api.us-west-2.amazonaws.com/prod/
ServiceTestStack.lambdaMainhelloworldapiEndpointE253830C = https://[redacted].execute-api.us-west-2.amazonaws.com/prod/
ServiceTestStack.lambdaMemoryhelloworldapimemoryEndpoint590B5394 = https://[redacted].execute-api.us-west-2.amazonaws.com/prod/

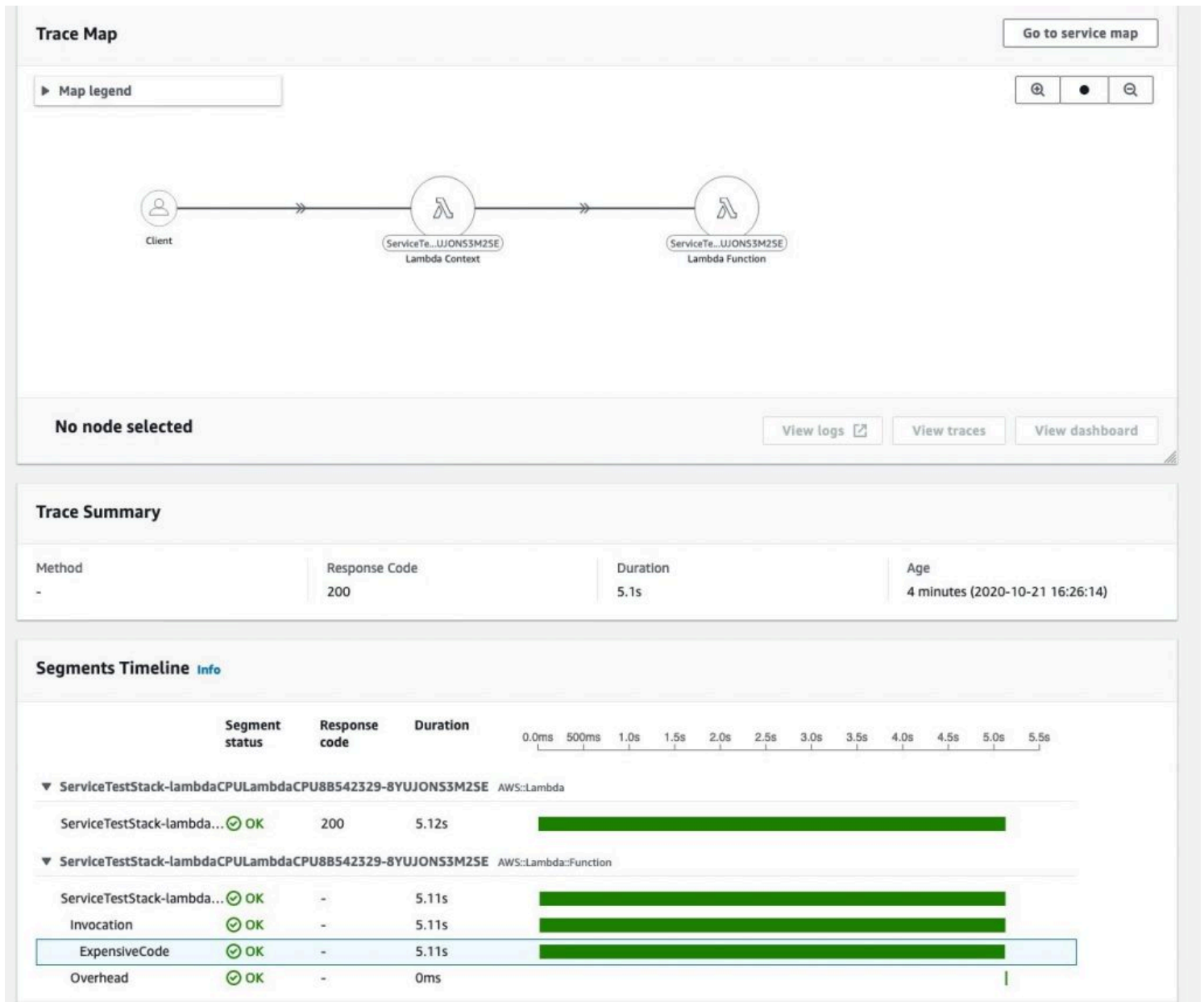
Stack ARN:
arn:aws:cloudformation:us-west-2:[redacted]:stack/ServiceTestStack/cf251220-0fe6-11eb-bbce-0219e54cc863

```

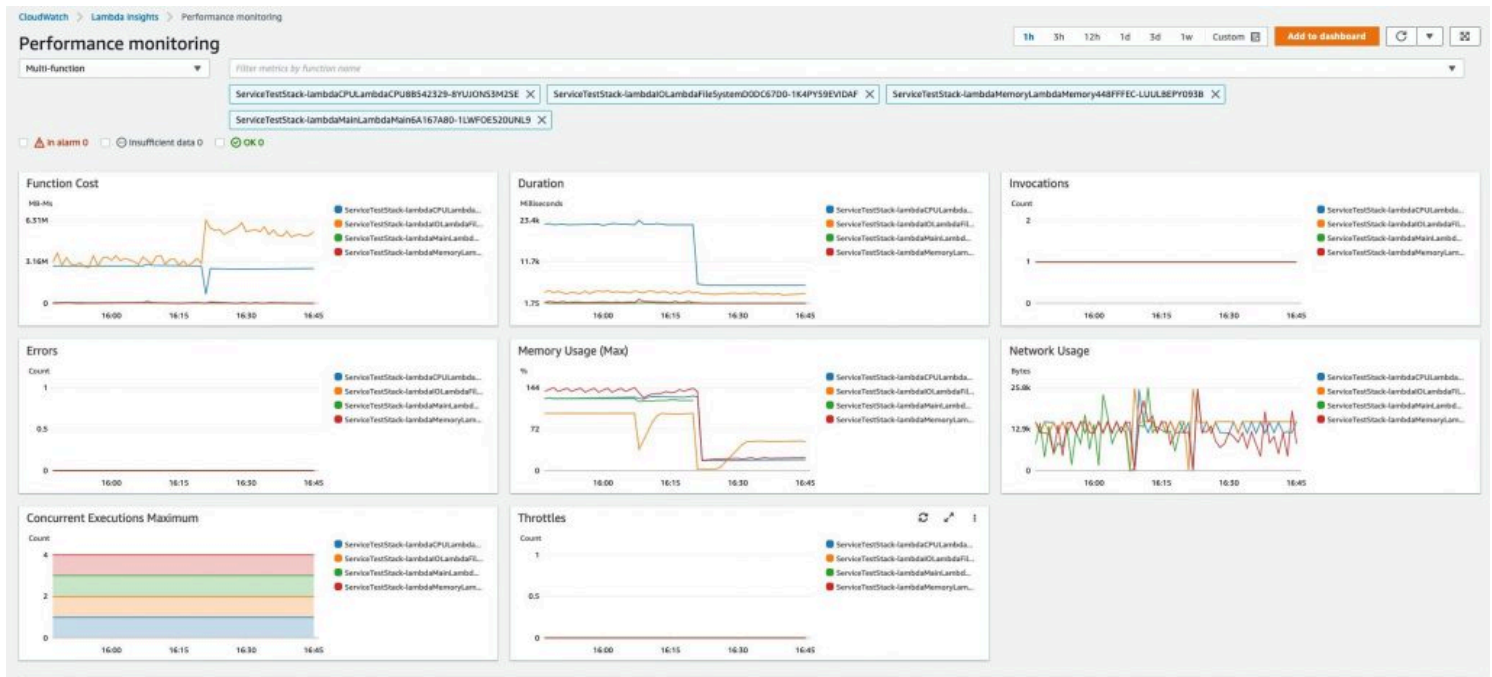
Opening Lambda Insights for the function again shows that both the “Duration” and “Memory Usage” charts have dropped significantly:



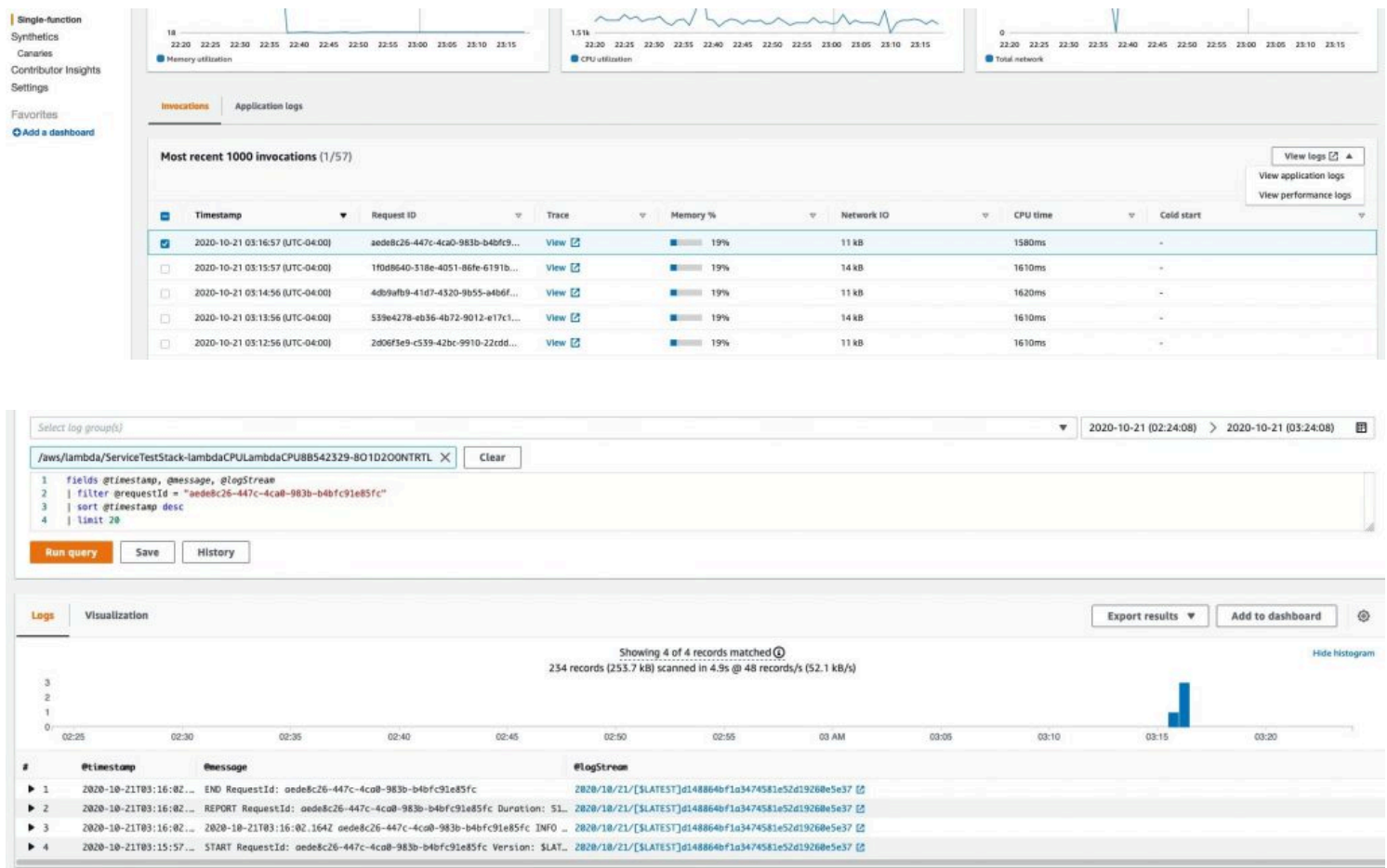
Viewing a trace shows the improved numbers for the AWS X-Ray segment running expensive code:



You can perform a similar rightsizing exercise for the other 3 Lambda functions deployed by the AWS CDK. Let's change the memory allocated to "LambdaMain" and "LambdaMemory" to 512 MB. For the "LambdaIO" function, we will change the memory to 2048 MB.

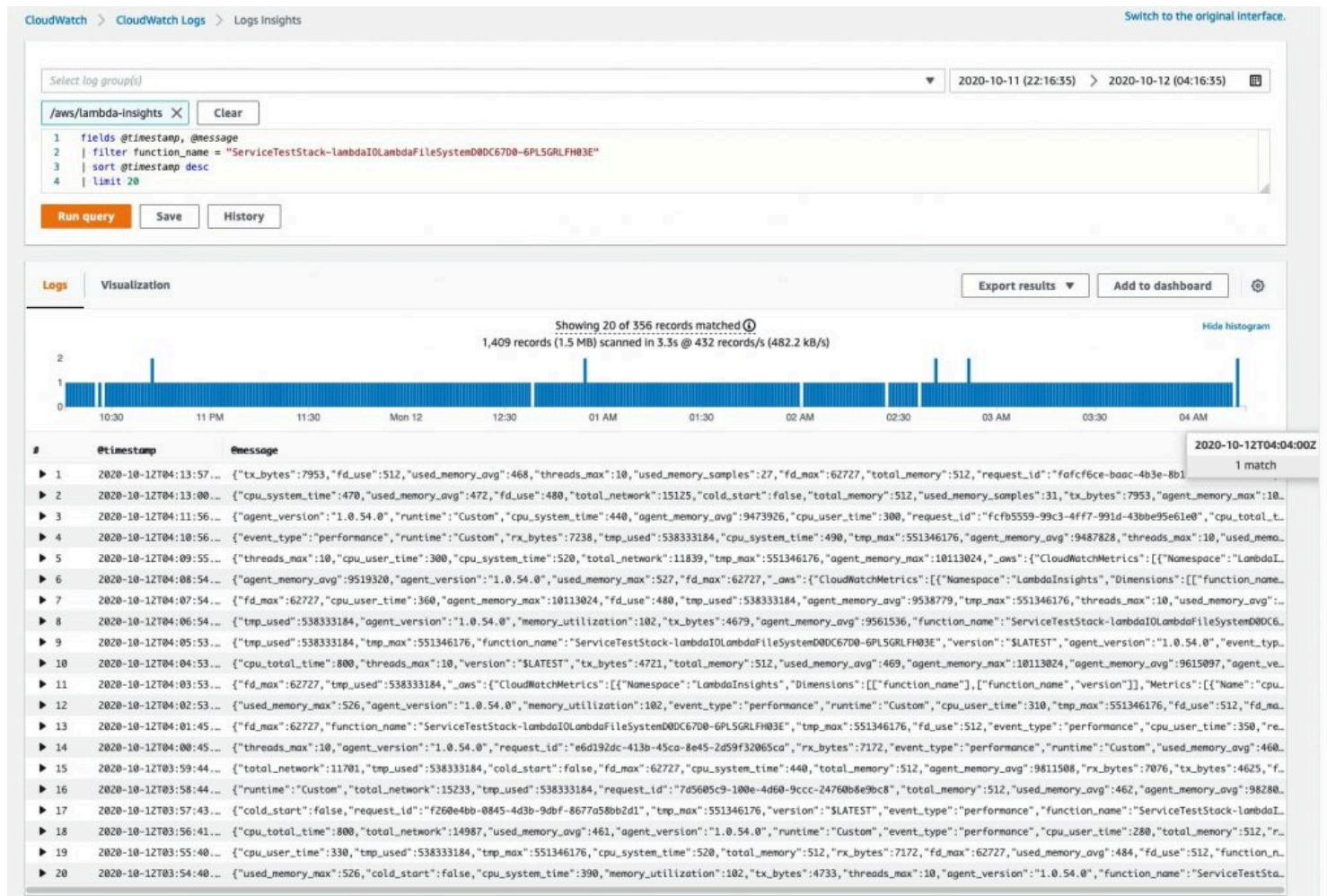


We can also look at the Lambda logs for a particular event by selecting an invocation and going to View Logs → View Application Logs. This will take us to CloudWatch Log Insights:



Selecting View Logs → Performance Logs will again take us to CloudWatch Log Insights for Log Group `/aws/lambda-insights`. Here we can see the performance logs of the Lambda function in the Embedded Metric

Format (EMF):



Cleanup

Perform the following steps to delete the resources created in this blog post:

1. Run "cdk destroy" from the terminal or AWS Cloud9 shell.
2. Delete the canaries in CloudWatch Synthetics.

Conclusion

In this blog post, we saw how we can use Lambda Insights to monitor and troubleshoot serverless applications based on Lambda on AWS. To learn more about AWS observability functionalities on Amazon CloudWatch and AWS X-Ray, see [One Observability Demo](#) workshop.

About the Author

Amit Kalawat is a Senior Solutions Architect at Amazon Web Services based out of New York. He works with enterprise customers as they transform their business and journey to the cloud.



TAGS: [Amazon CloudWatch](#), [AWS Lambda](#)