sumo logic

Observability Fundamentals

Student Lab Guide

Rev 08.08.2024 K

Table of Labs

Lab 0: Log in to the training environment

You'll need **two different browser** windows open for the lab activities. In one browser, you will be logged in as yourself, in **your own account**. In another browser, you'll be logged into a **Sumo Logic training lab environment**.

The training lab environment is separate from your other accounts. So, you'll need to create another login. To access the training lab environment:

1. Open a new window in a different browser or incognito window.
2. Navigate to https://service.sumologic.com in the new browser window.
3. Choose a number between 001 and 999. Remember this number, since you'll use it in all your labs.
4. Enter **training+analyst###@sumologic.com** in the **Email** field. Replace **###** with the number you chose in Step 3.
5. Enter the **Password** provided to you by your instructor.

Note: The password changes monthly. You can find the password on the training homepage for more information.

If you're reading this on Chrome, open a Firefox window. Using separate browsers will keep you logged in to your regular Sumo Logic account and the training lab environment at the same time. If you don't have two different browsers on your machine, you can open a Private or Incognito window instead.

In one browser, log in to **your own account**. Use this browser to navigate the lab activities, and take the exam. This way, you'll get credit for taking the course and passing the exam. Make sure you're logged in as yourself by clicking the profile icon in the upper right corner.

In another browser, log in to a **Sumo Logic training lab environment**. You can explore this environment freely, without impacting your dev, prod, or trial environments. Our lab activities are designed for the training lab environment, and may not work in other Sumo Logic accounts.

Note: The training accounts are public. Be careful what kind of personal information you share, like your real name or email address. These training accounts are wiped clean weekly, so make sure you complete any lab activities and save any data you want to keep, as you may not be able to recover it later.

Lab 1: Finding the root cause of latency using Log Search

In this lab, you'll learn how dashboards created based on metrics are instrumental in identifying the root cause of an outage. Once you've identified a potential latency issue, you'll investigate it further using Log Search.

Note: All labs assume you're using a **training+analyst###@sumologic.com** account. The data you see may vary depending on your environment if you're using your own credentials or a Sumo Logic trial account instead of a training account. To access a training account, review **Lab 0**.

Here's the scenario you are trying to troubleshoot and isolate the root cause.

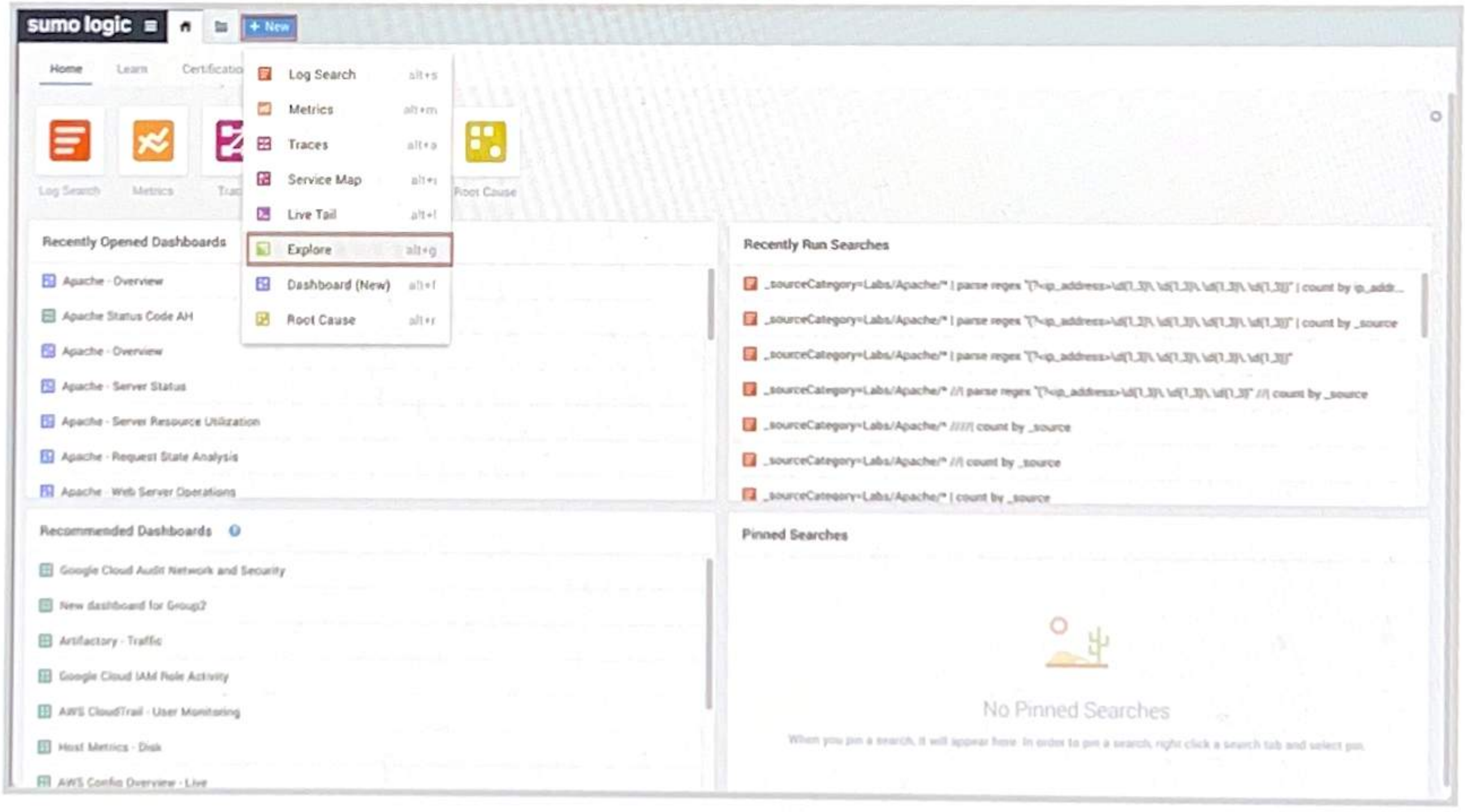
You, a DevOps specialist for your company, are supporting an app on AWS that uses following AWS services:

- ELB (Application Load Balancer)
- Host/EC2
- DynamoDB

Your company app is reporting unusual latency in loading the app web page, along with occasional outages in the US area. You need to find out the root cause of the latency and outages ASAP.

The lab begins:

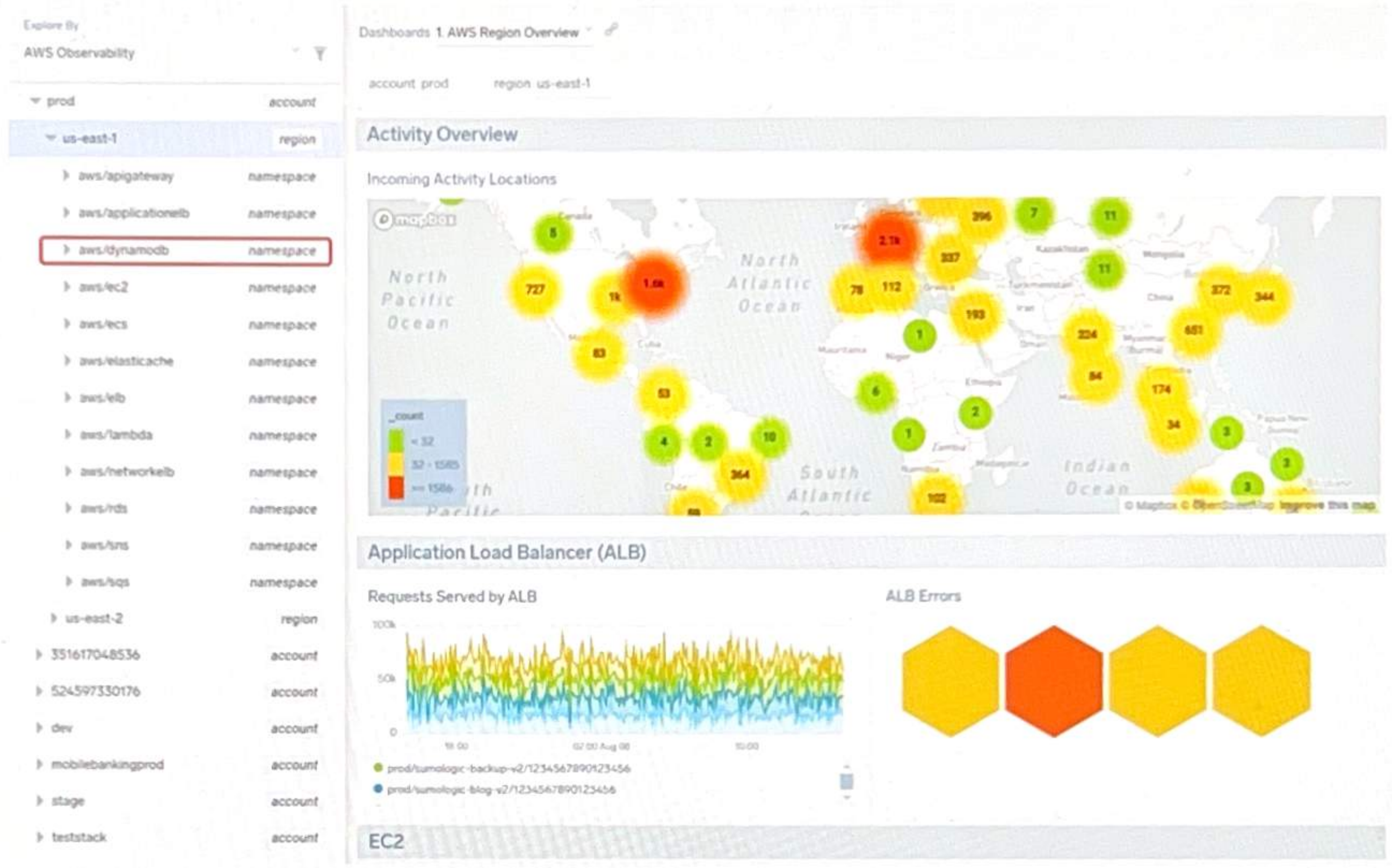
1. In the Sumo Logic UI, near the top of the Sumo Logic UI, click **+New > Explore**.



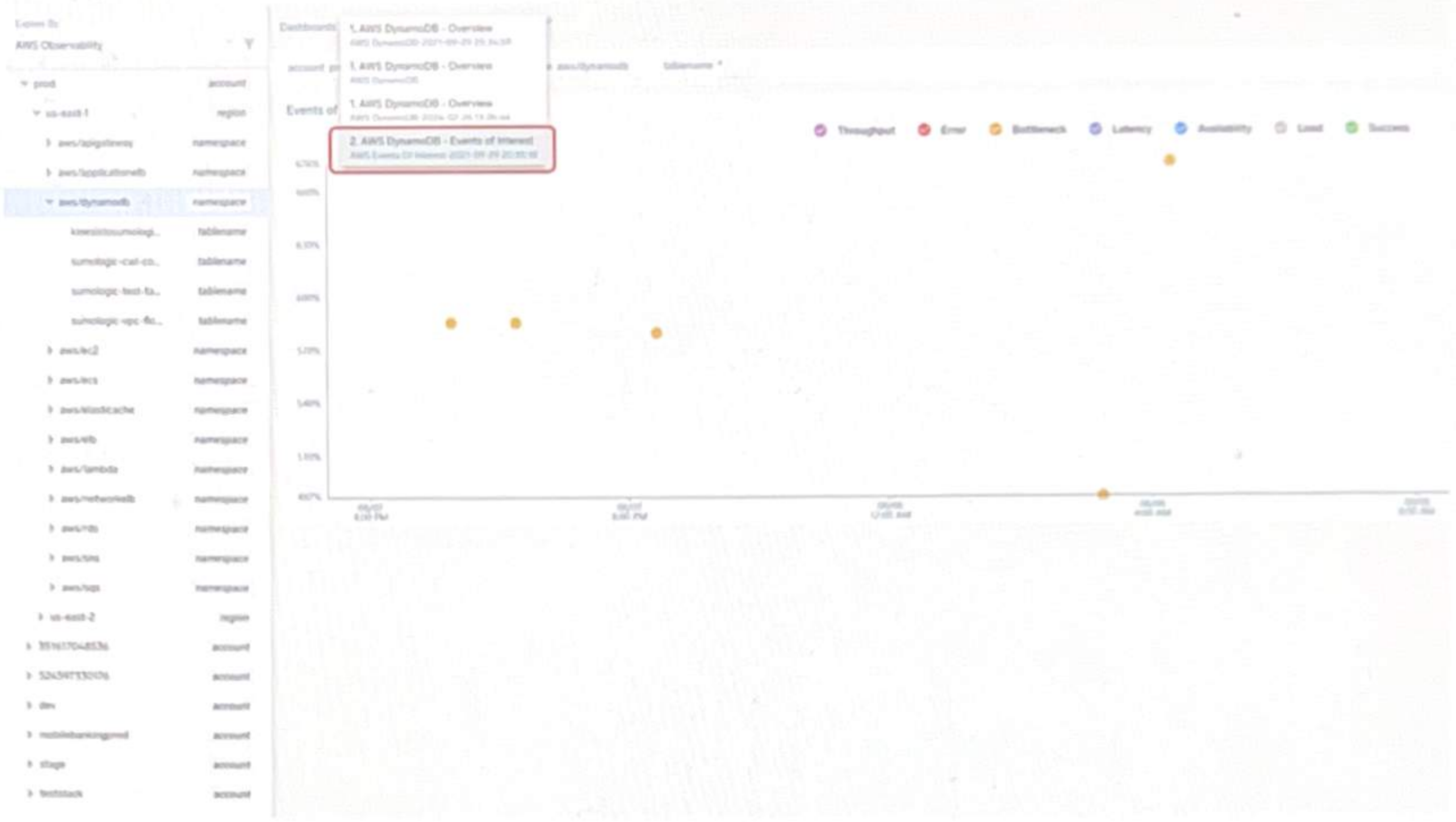
```
_sourceCategory=Labs/Apache/* | parse regex "(?<ip_address>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})" | count by ip_addr...
_sourceCategory=Labs/Apache/* | parse regex "(?<ip_address>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})" | count by _source
_sourceCategory=Labs/Apache/* | parse regex "(?<ip_address>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})"
_sourceCategory=Labs/Apache/* //| parse regex "(?<ip_address>\d{1,3}\.\d{1,3}\.\d{1,3}"//| count by _source
_sourceCategory=Labs/Apache/* /////| count by _source
_sourceCategory=Labs/Apache/* //| count by _source
_sourceCategory=Labs/Apache/* | count by _source
```

2. In the top left corner of the resulting screen, click **Explore by**, and select **AWS Observability**.

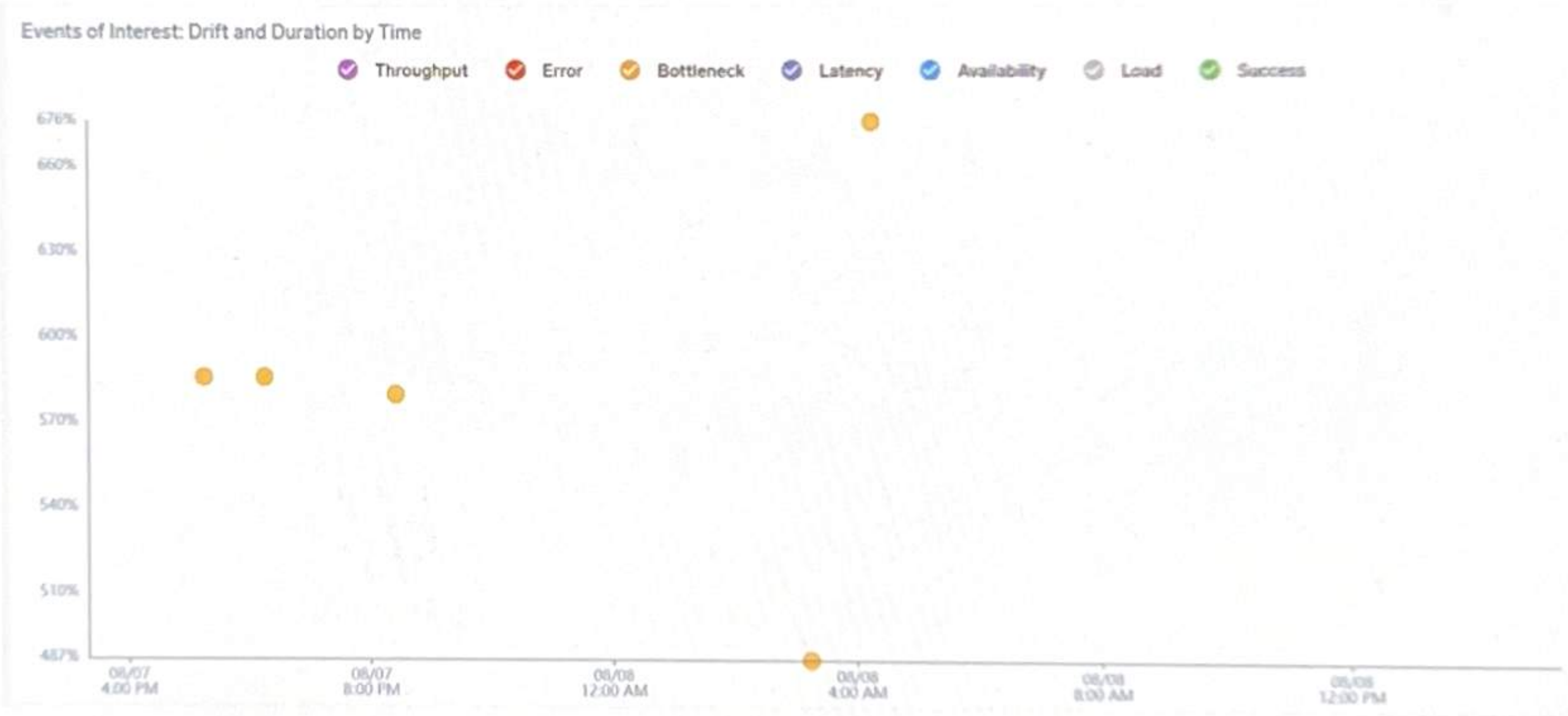
3. In the left pane, click the **us-east-1** region under the **Prod** account.
4. You will see a number of separate AWS services listed under **us-east-1.** Locate and click the entry for **aws/dynamodb**



5. In the right pane, in the Dashboards dropdown on top, select **AWS DynamoDB - Events of Interest**.

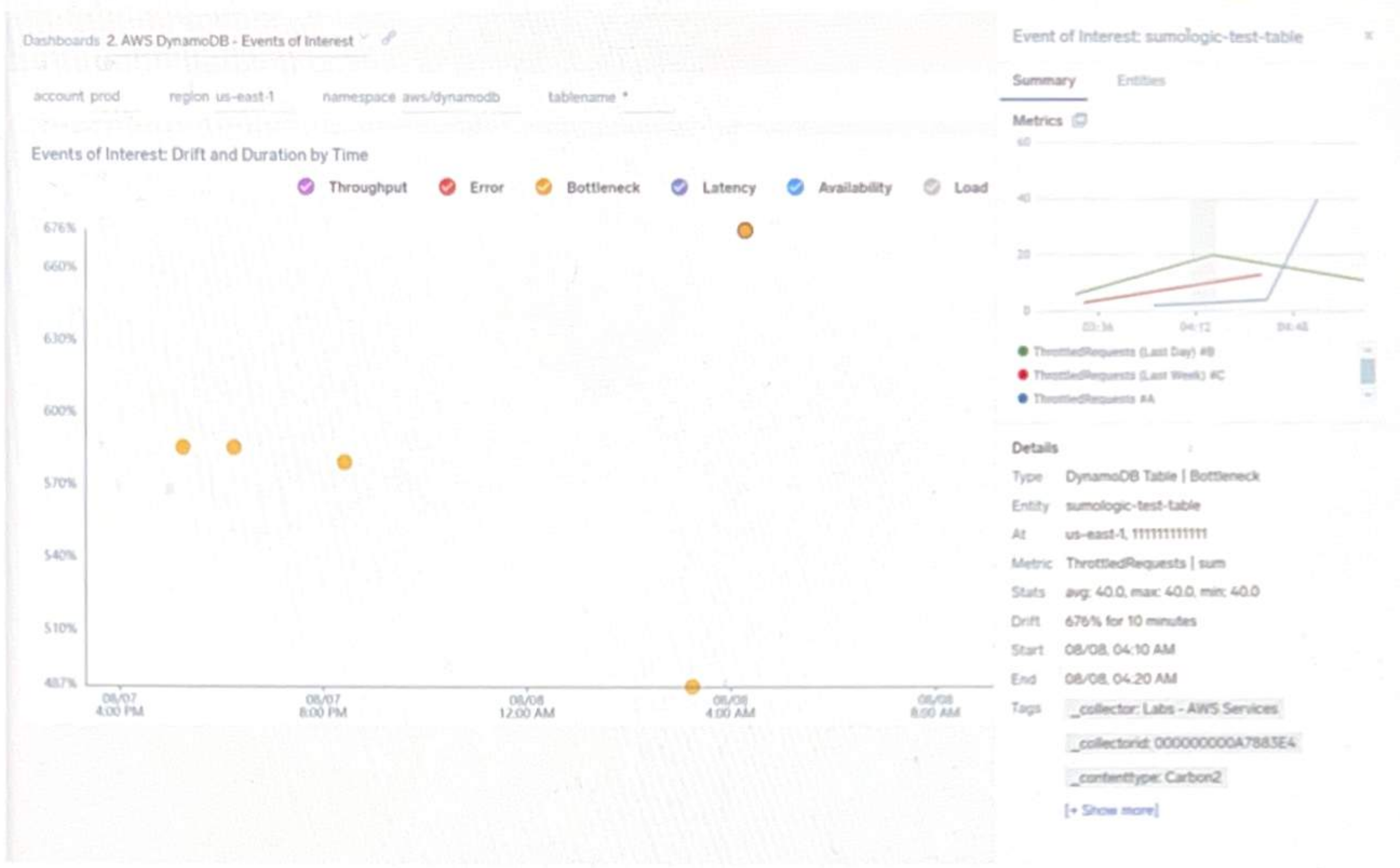


6. In this view you'll see a graph displaying “events of interest” that have been recorded in the Dynamo DB, such as anomalous errors, latency spikes, or bottlenecks. You should see one or more yellow circles representing detected bottlenecks in DB processing.



The X-axis represents time, and the Y-axis represents the “severity” of the event (the percentage increase between the amount of errors recorded at that time versus “normal” operation).

7. Click on any of the yellow circles and open up the right sidebar window with more information.

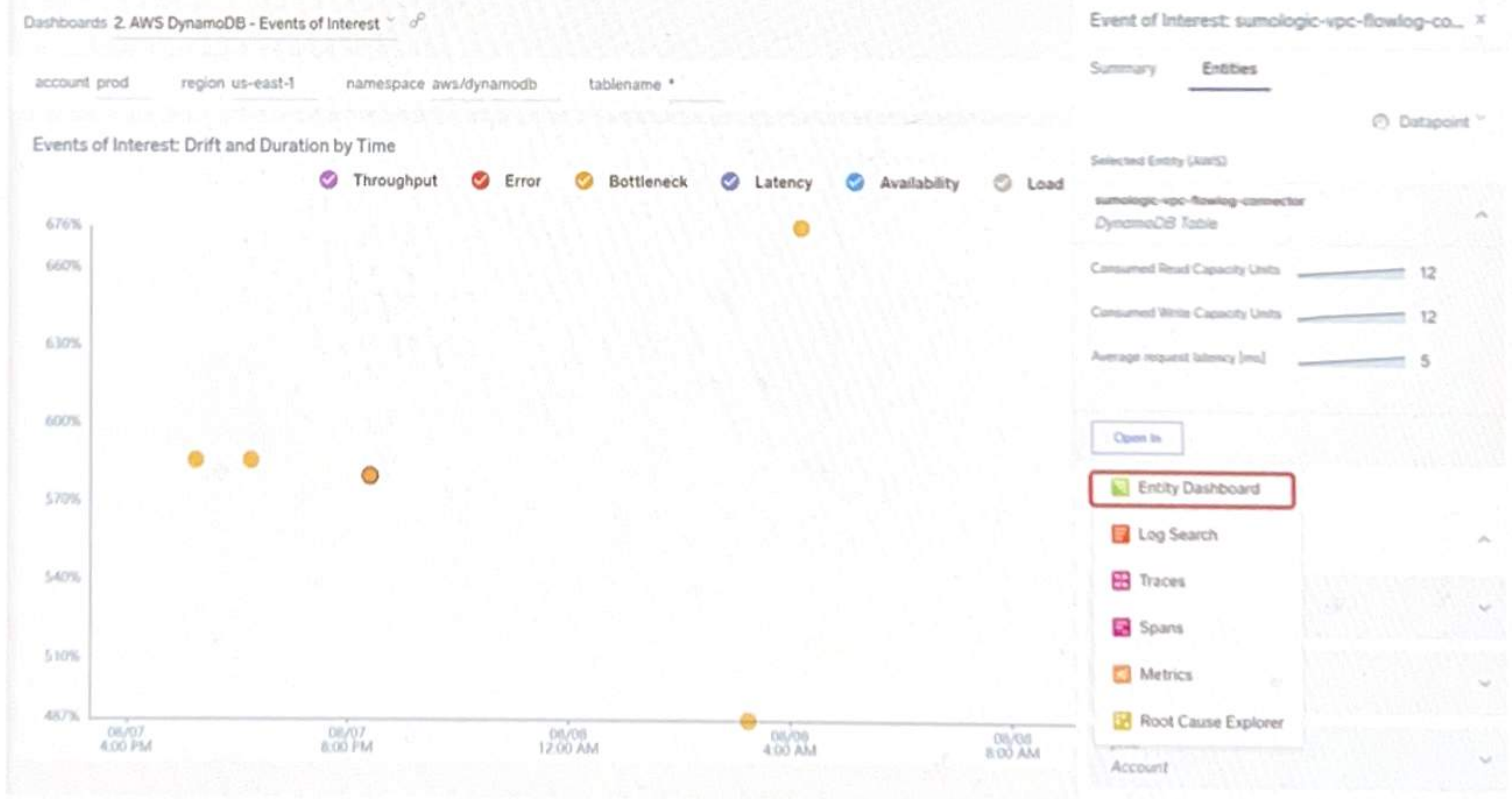


Details

Type	DynamoDB Table | Bottleneck
Entity	sumologic-test-table
At	us-east-1, 111111111111
Metric	ThrottledRequests | sum
Stats	avg: 40.0, max: 40.0, min: 40.0
Drift	676% for 10 minutes
Start	08/08, 04:10 AM
End	08/08, 04:20 AM
Tags	_collector: Labs - AWS Services
	_collectorid: 000000000A7883E4
	_contenttype: Carbon2

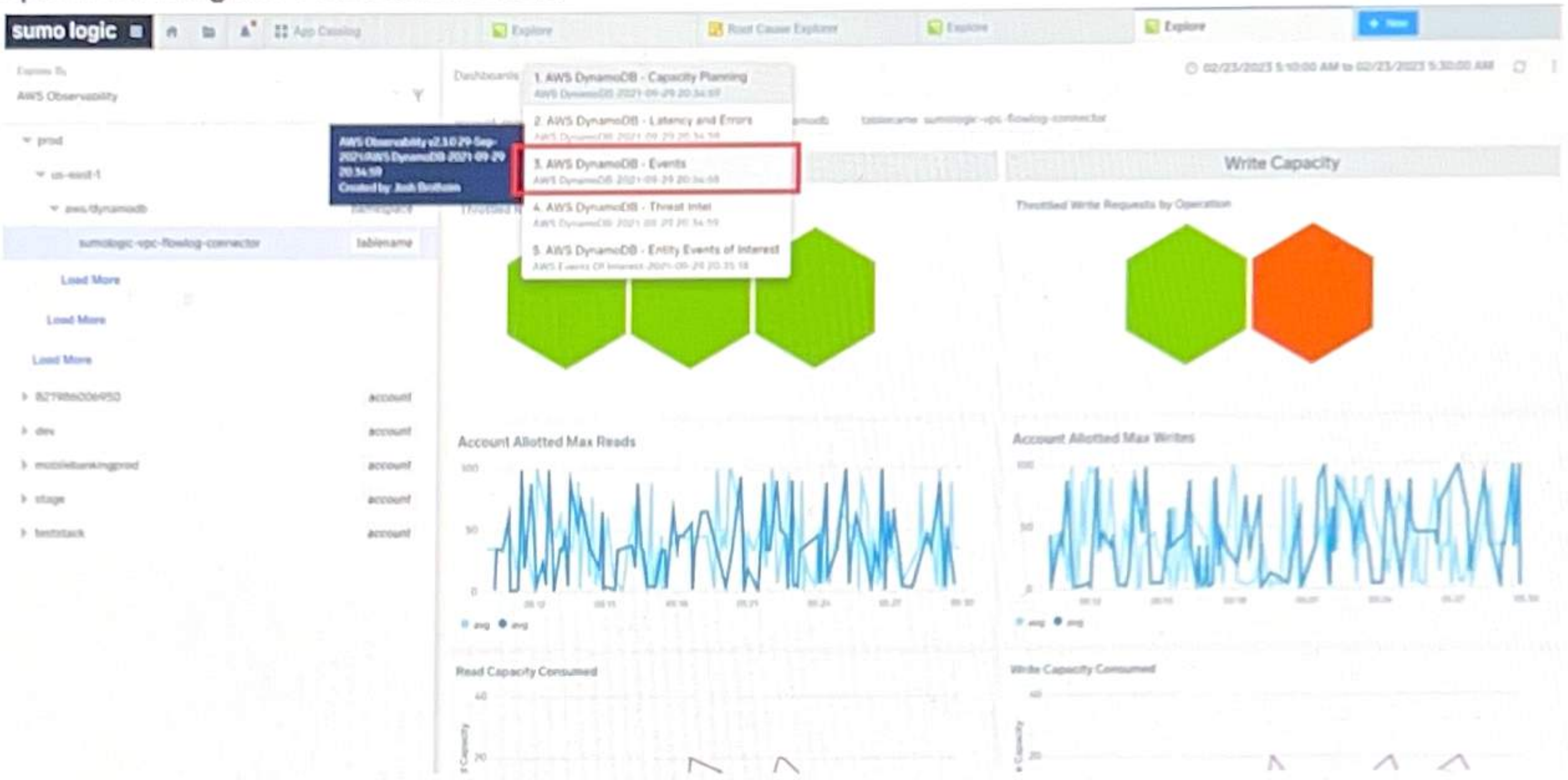
8. The information here indicates that DB requests are being throttled for some reason. Now we want to dig into how and why this is happening. Click on the **Entities** tab in the right sidebar.

9. Then click the **Open In** button and select **Entity Dashboard**.

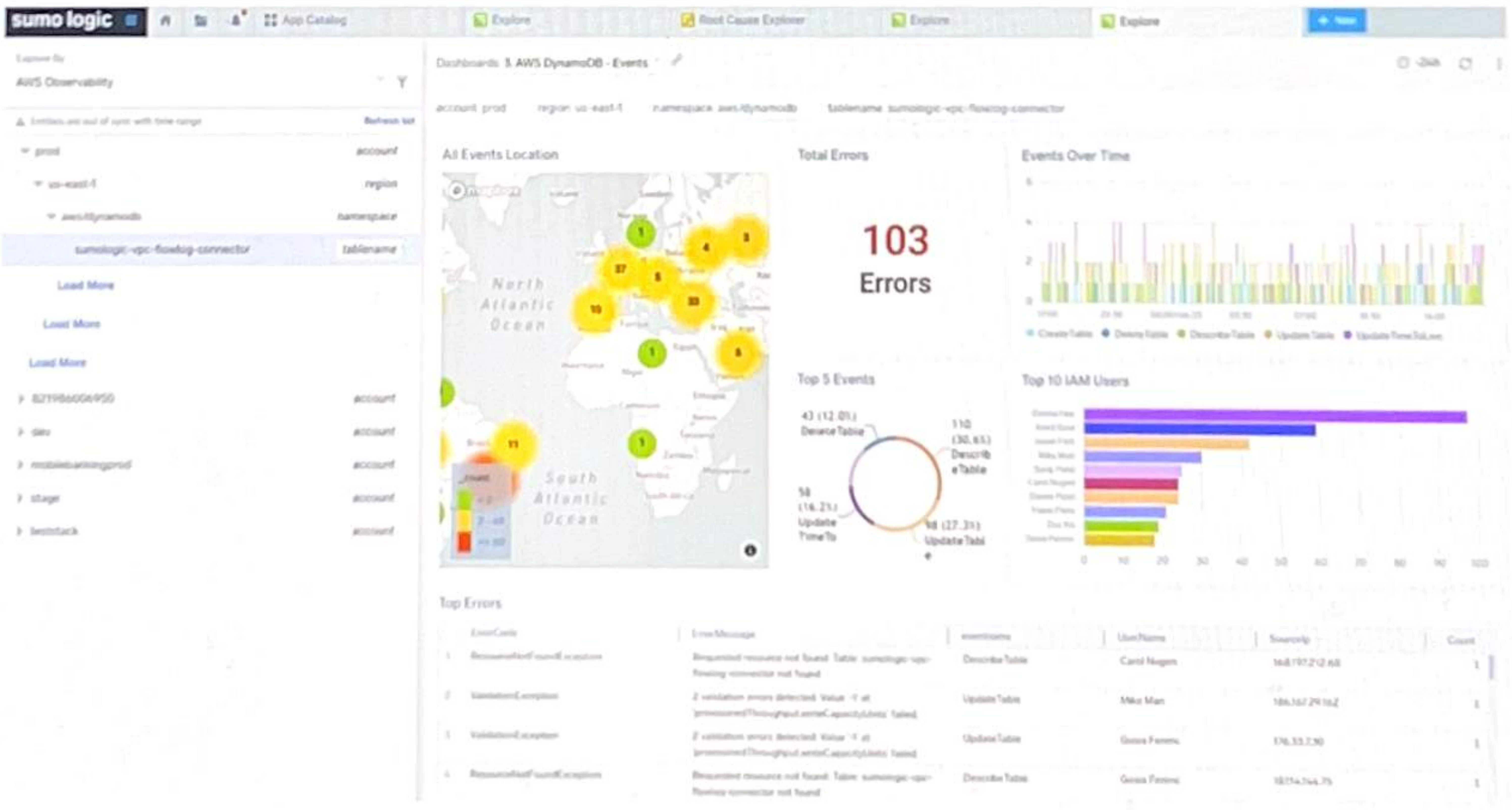


Consumed Read Capacity Units	12
Consumed Write Capacity Units	12
Average request latency [ms]	5

10. On the following screen, open the relevant DynamoDB Events dashboard by clicking the **AWS DynamoDB - Events** option in the dropdown. The data in the dashboard will be updated using the relevant metrics.


1. AWS DynamoDB - Capacity Planning
2. AWS DynamoDB - Latency and Errors
3. AWS DynamoDB - Events
4. AWS DynamoDB - Threat Intel
5. AWS DynamoDB - Entity Events of Interest
Write Capacity
Throttled Write Requests by Operation
Account Allotted Max Reads
Account Allotted Max Writes
Read Capacity Consumed
Write Capacity Consumed
Load More
account

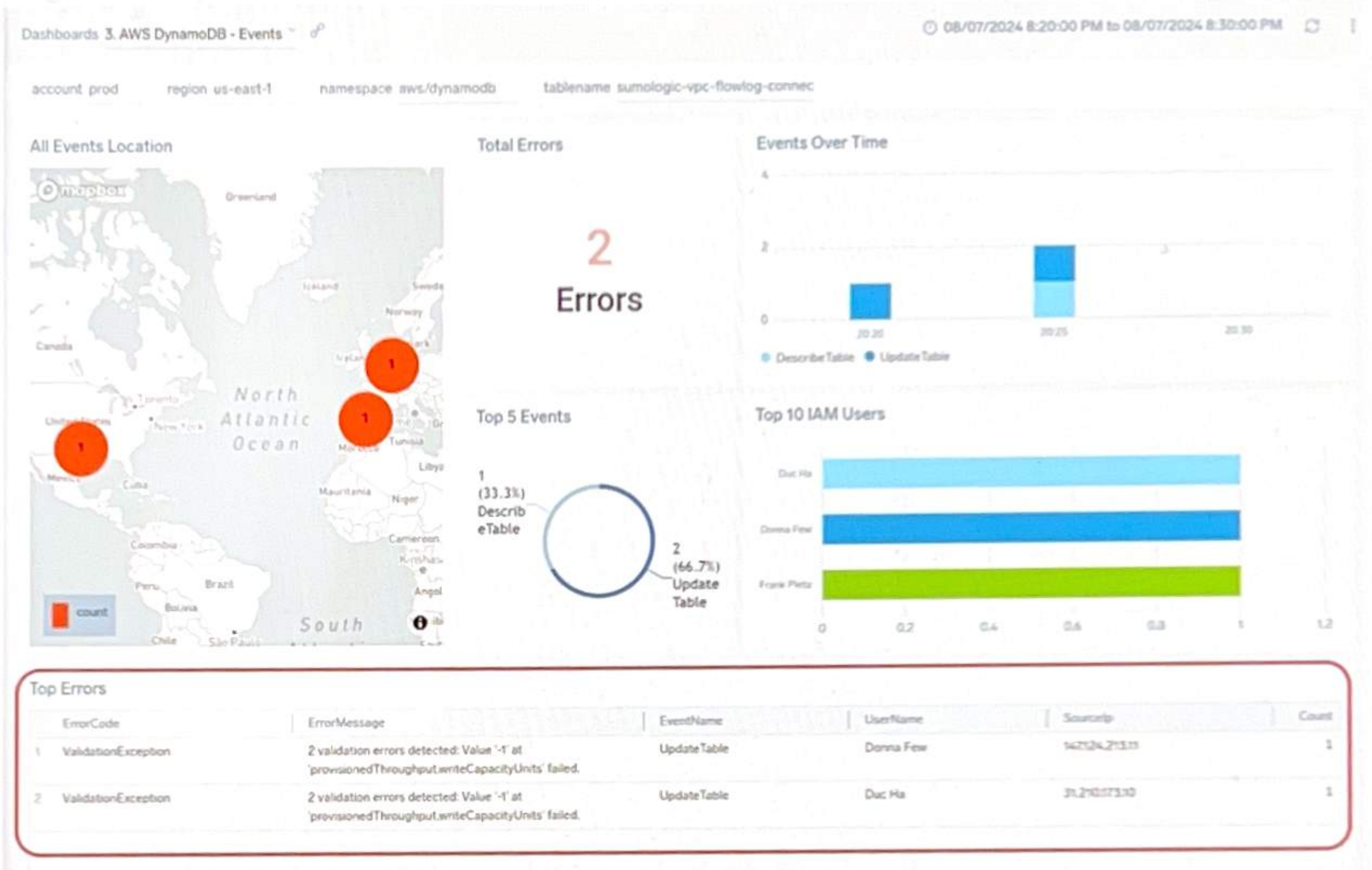
11. The dashboard displays a number of panels including No. of Errors, Top 5 Events, and Events over Time. You can see the total number of errors and also the associated events that occurred in the DynamoDB. (Note that you can change the time span for the dashboard in the upper right to expand or contract the time range to see how the metrics and displays change accordingly).



Top Errors

	ErrorCode	ErrorMessage	eventname	UserName	SourceIp	Count
1	ResourceNotFoundException	Requested resource not found: Table: sumologic-vpc-flowlog-connector not found	DescribeTable	Carol Nugen	[illegible]	1
2	ValidationException	2 validation errors detected: Value '-1' at 'provisionedThroughput.writeCapacityUnits' failed	UpdateTable	Mike Man	[illegible]	1
3	ValidationException	2 validation errors detected: Value '-1' at 'provisionedThroughput.writeCapacityUnits' failed	UpdateTable	Gosia Ferenc	[illegible]	1
4	ResourceNotFoundException	Requested resource not found: Table: sumologic-vpc-flowlog-connector not found	DescribeTable	Gosia Ferenc	[illegible]	1

12. Once you find the relevant data and metrics in the events dashboard, you can drill down further into the associated logs. Scroll down to see the **Top Errors** table for clues as to the root cause of the DB throttling.

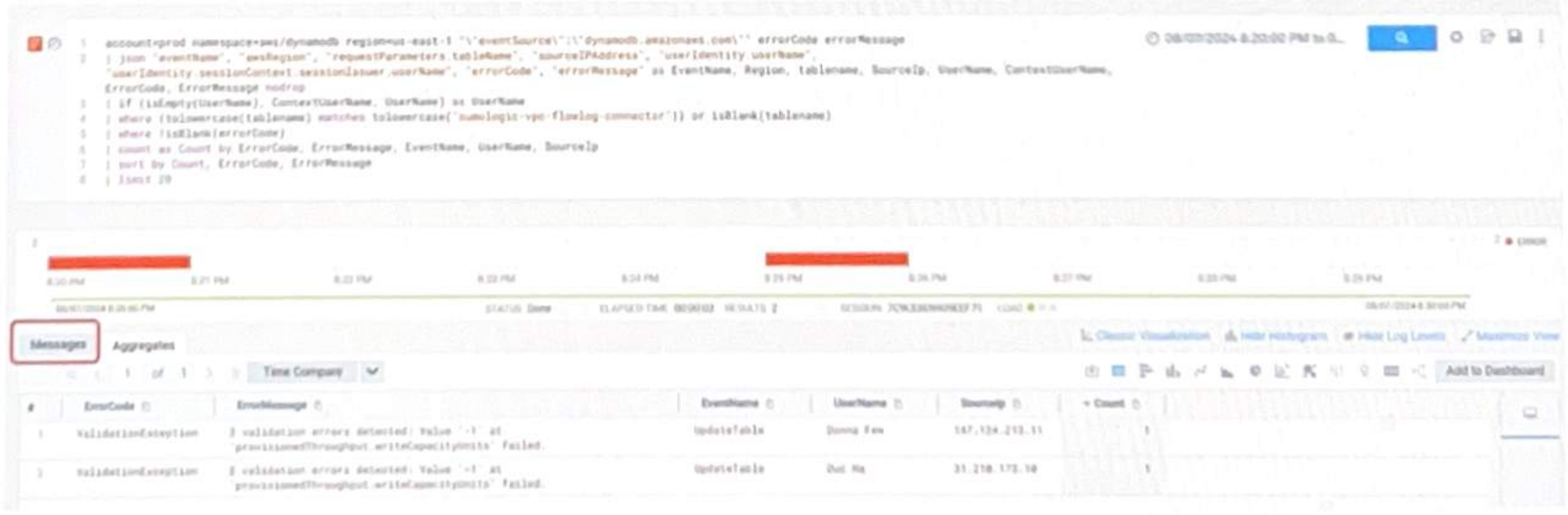


Top Errors

	ErrorCode	ErrorMessage	EventName	UserName	SourceIp	Count
1	ValidationException	2 validation errors detected: Value '-1' at 'provisionedThroughput.writeCapacityUnits' failed.	UpdateTable	Donna Few	147.124.213.11	1
2	ValidationException	2 validation errors detected: Value '-1' at 'provisionedThroughput.writeCapacityUnits' failed.	UpdateTable	Duc Ha	31.210.173.10	1

13. Click the "three dot" icon in the top right corner of the **Top Errors** table, then select **Open in Log Search** to open the relevant logs.

14. When you open the event logs in Search mode, you will first see the **aggregate** events, where you see someone has updated the table. Click the **Messages** tab to view the individual logs.

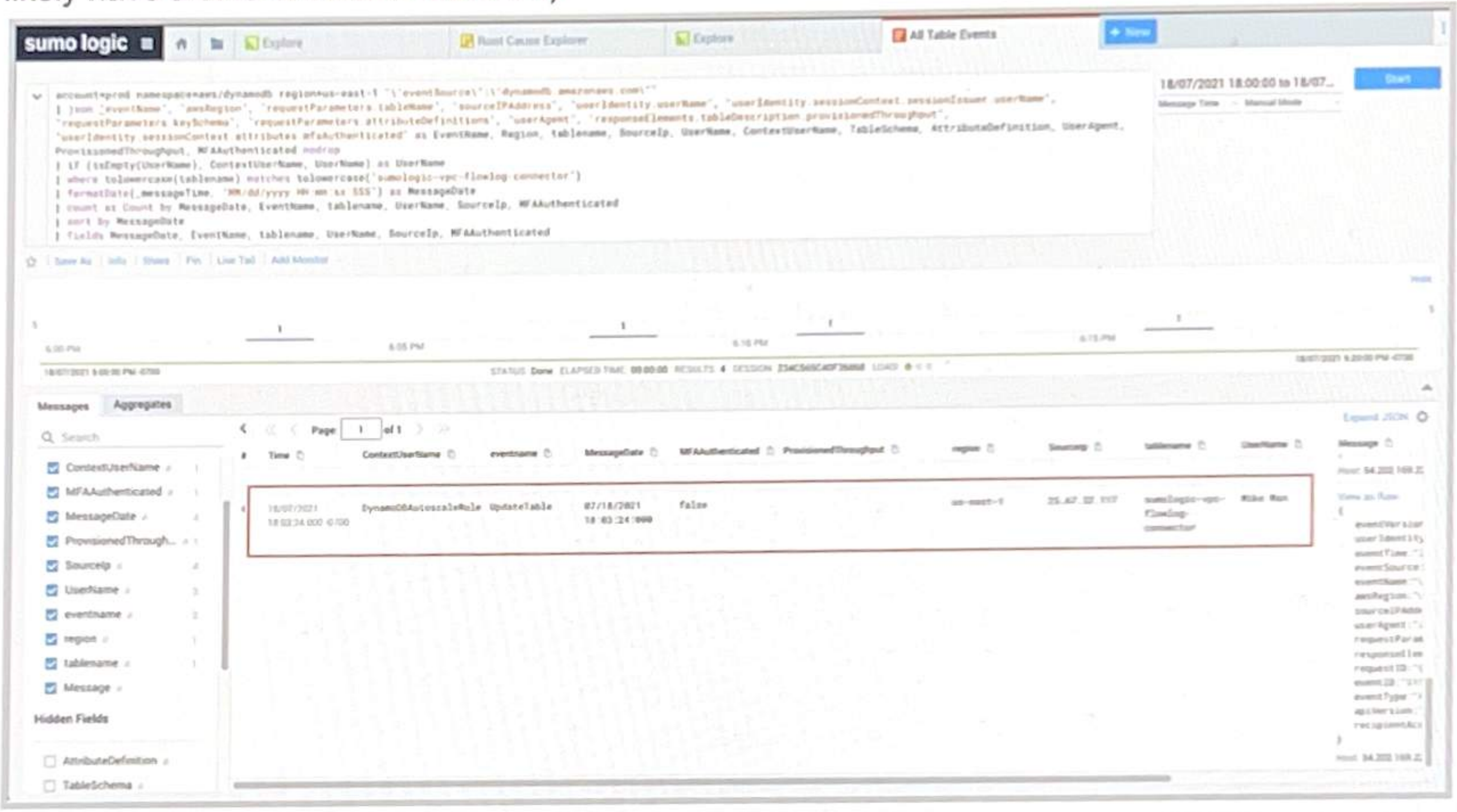


```
account=prod namespace=aws/dynamodb region=us-east-1 "\"eventSource\":\"dynamodb.amazonaws.com\"" errorCode errorMessage
| json "eventName", "awsRegion", "requestParameters.tableName", "sourceIPAddress", "userIdentity.userName",
"userIdentity.sessionContext.sessionIssuer.userName", "errorCode", "errorMessage" as EventName, Region, tablename, SourceIp, UserName, ContextUserName,
ErrorCode, ErrorMessage nodrop
| if (isEmpty(UserName), ContextUserName, UserName) as UserName
| where (tolowercase(tablename) matches tolowercase("sumologic-vpc-flowlog-connector")) or isBlank(tablename)
| where !isBlank(errorCode)
| count as Count by ErrorCode, ErrorMessage, EventName, UserName, SourceIp
| sort by Count, ErrorCode, ErrorMessage
| limit 20
```

#	ErrorCode	ErrorMessage	EventName	UserName	SourceIp	Count
1	ValidationException	2 validation errors detected: Value '-1' at 'provisionedThroughput.writeCapacityUnits' failed.	UpdateTable	Donna Few	147.124.213.11	1
2	ValidationException	2 validation errors detected: Value '-1' at 'provisionedThroughput.writeCapacityUnits' failed.	UpdateTable	Duc Ha	31.210.173.10	1

15. Now all the relevant individual log entries are displayed. If desired, you can apply filters in the left pane to focus on specific fields to better search through the events you need to investigate. For instance, try deselecting the fields **'ContextUserName'** or **'ErrorCode'** with the checkbox and see how the data updates.

16. When you go through the resulting update table log events, you find that someone named **Mike Man** called UpdateTable and changed the read/write parameter (IOPS). (NOTE that the names in these training log entries are randomized so your log entry will likely have a different name attached)



```
account=prod namespace=aws/dynamodb region=us-east-1 "\"eventSource\":\"dynamodb.amazonaws.com\""
| json "eventName", "awsRegion", "requestParameters.tableName", "sourceIPAddress", "userIdentity.userName", "userIdentity.sessionContext.sessionIssuer.userName",
"requestParameters.keySchema", "requestParameters.attributeDefinitions", "userAgent", "responseElements.tableDescription.provisionedThroughput",
"userIdentity.sessionContext.attributes.mfaAuthenticated" as EventName, Region, tablename, SourceIp, UserName, ContextUserName, TableSchema, AttributeDefinition, UserAgent,
ProvisionedThroughput, MFAAuthenticated nodrop
| if (isEmpty(UserName), ContextUserName, UserName) as UserName
| where tolowercase(tablename) matches tolowercase("sumologic-vpc-flowlog-connector")
| formatDate(_messageTime, "MM/dd/yyyy HH:mm:ss:SSS") as MessageDate
| count as Count by MessageDate, EventName, tablename, UserName, SourceIp, MFAAuthenticated
| sort by MessageDate
| fields MessageDate, EventName, tablename, UserName, SourceIp, MFAAuthenticated
```

STATUS Done ELAPSED TIME 00:00:00 RESULTS 4

#	Time	ContextUserName	eventname	MessageDate	MFAAuthenticated	ProvisionedThroughput	region	SourceIp	tablename	UserName
4	18/07/2021 18:03:24.000 -0700	DynamoDBAutoscaleRole	UpdateTable	07/18/2021 18:03:24:000	false		us-east-1	25.47.22.117	sumologic-vpc-flowlog-connector	Mike Man

This developer had evidently reconfigured DynamoDB to use lower-provisioned IOPS (Input/Output Operations Per Second) which caused throttling and the subsequent outages. This will give your organization the clue to reconfigure the IOPS and the throttling issue should be resolved.

Lab 2: When are Logs the best bet in Observability?

In this lab, you'll learn that when alerted to an incident, diving deep into the logs can help you troubleshoot and resolve an issue.

Note: All labs assume you're using a training+analyst###@sumologic.com account. The data you see may vary depending on your environment if you're using your own credentials or a Sumo Logic trial account instead of a training account. To access a training account, review **Lab 0**.

Here's the scenario you are trying to troubleshoot.

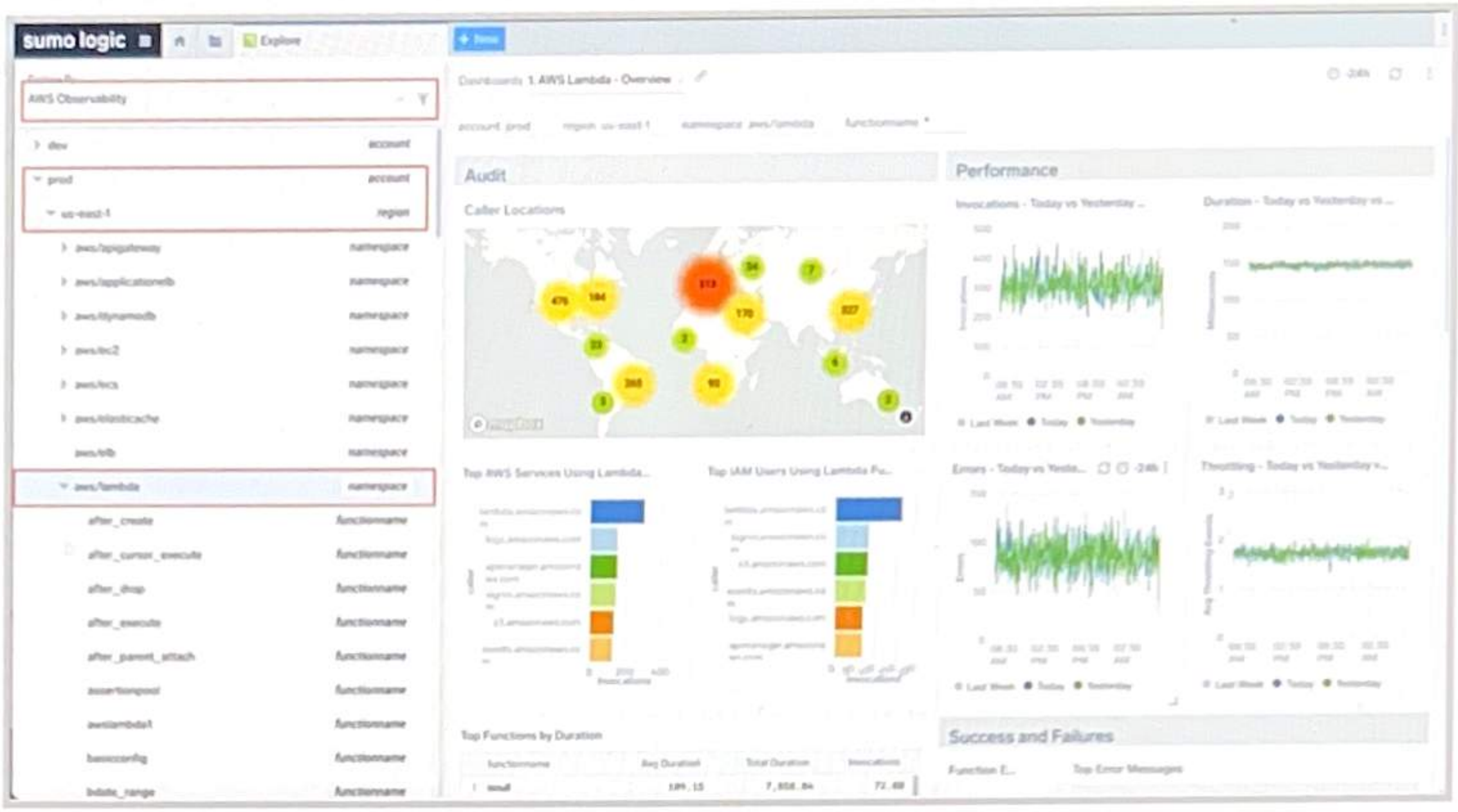
You are an SRE and you've got an alert on an incident. System end users on the **US east coast** have been experiencing an outage in our app in the **last 24 hours**. You've got an indication that the problem is on the **Lambda** server. Now you need to take a look to see when all of this started. What metrics are you seeing and how can you determine when this happened?

The lab begins.

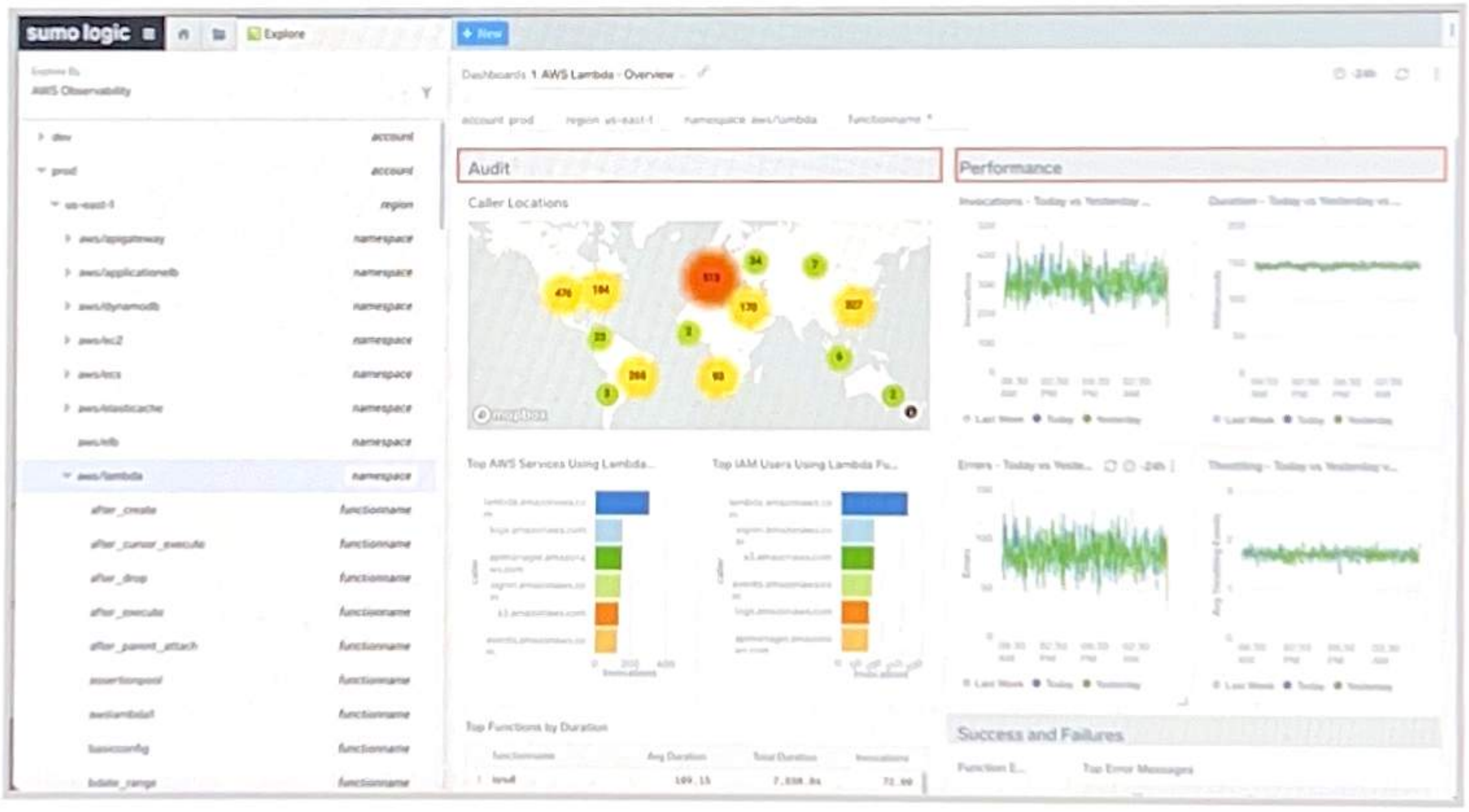
1. Near the top of the Sumo Logic UI, click **+New** > **Explore**.
   We know that the problem is in the Lambda server, so let's start there.

2. In the top left corner of the resulting screen, click **Explore by**, and select **AWS Observability**.

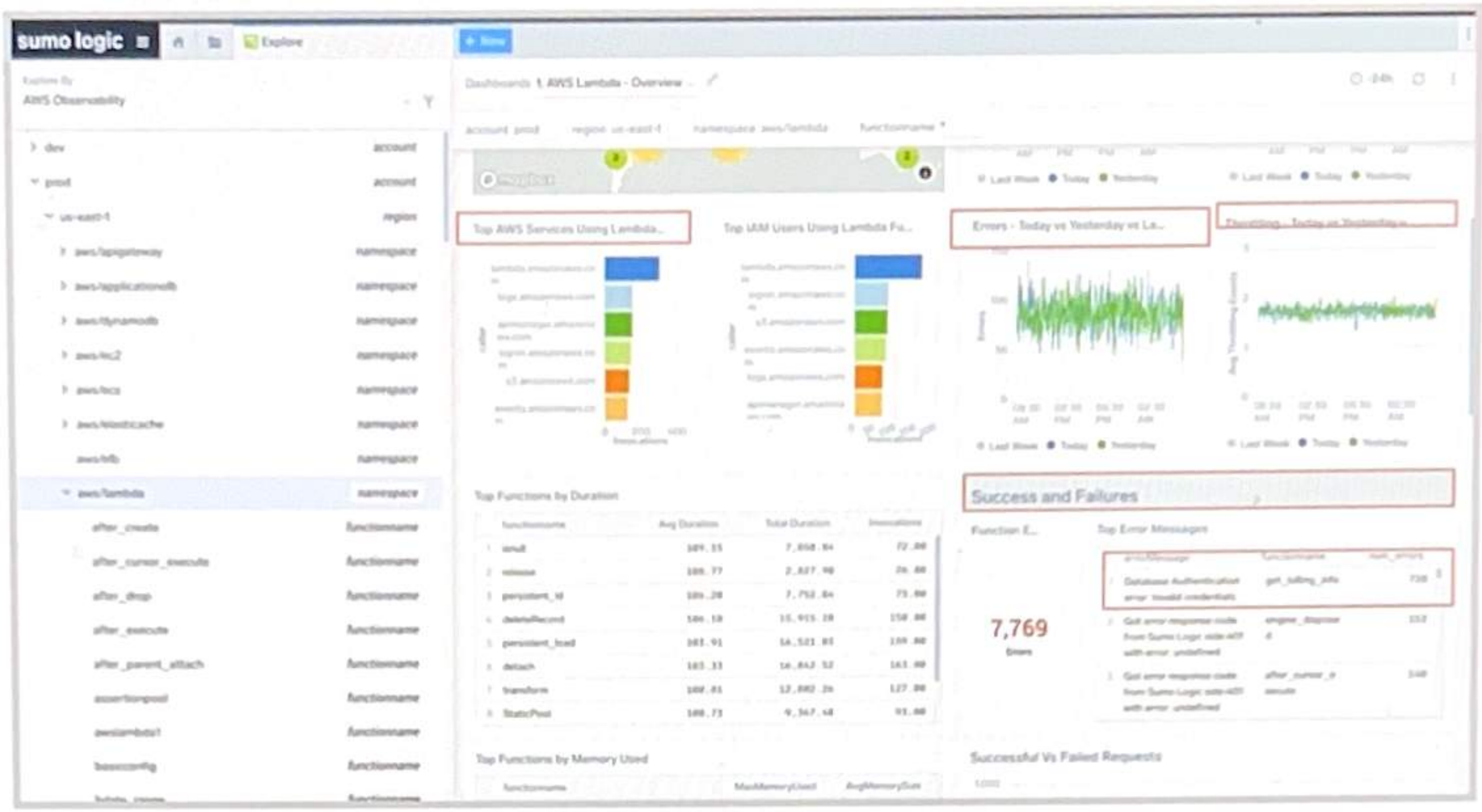
3. In the left pane, click the **us-east-1** region under the **Prod** account.



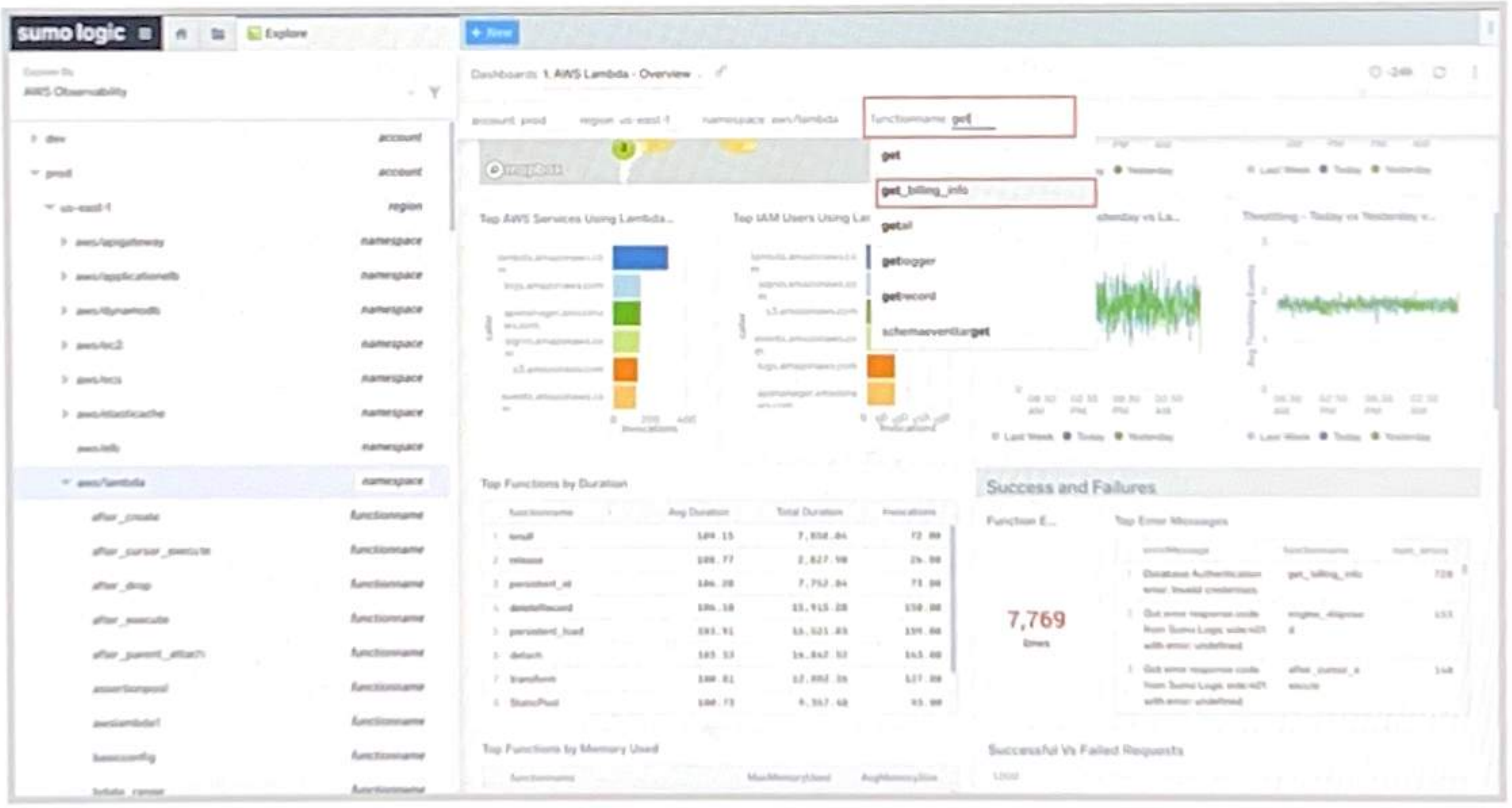
4. The **AWS Lambda - Overview** dashboard displays the **Audit** and **Performance** panels.



If you scroll down, you can examine the data to find out which functions are taking the longest to run along with which ones consume the highest amount of memory. Using the **Success and Failure** panel a little further down on the right, you can see that the **get_billing_info** function has recorded the highest number of errors in this time period. Sounds like a good lead to investigate!


sumo logic
Explore
AWS Observability
Dashboards 1. AWS Lambda - Overview
Top AWS Services Using Lambda...
Top IAM Users Using Lambda Fu...
Errors - Today vs Yesterday vs La...
Throttling - Today vs Yesterday v...
Top Functions by Duration
Success and Failures
Top Error Messages
7,769
Top Functions by Memory Used
Successful Vs Failed Requests

5. Apply a filter by selecting ‘get_billing_info’ in the **functionname** drop down on the top of the screen. Now we will see only the specific info related to calls to the ‘get_billing_info’ function.

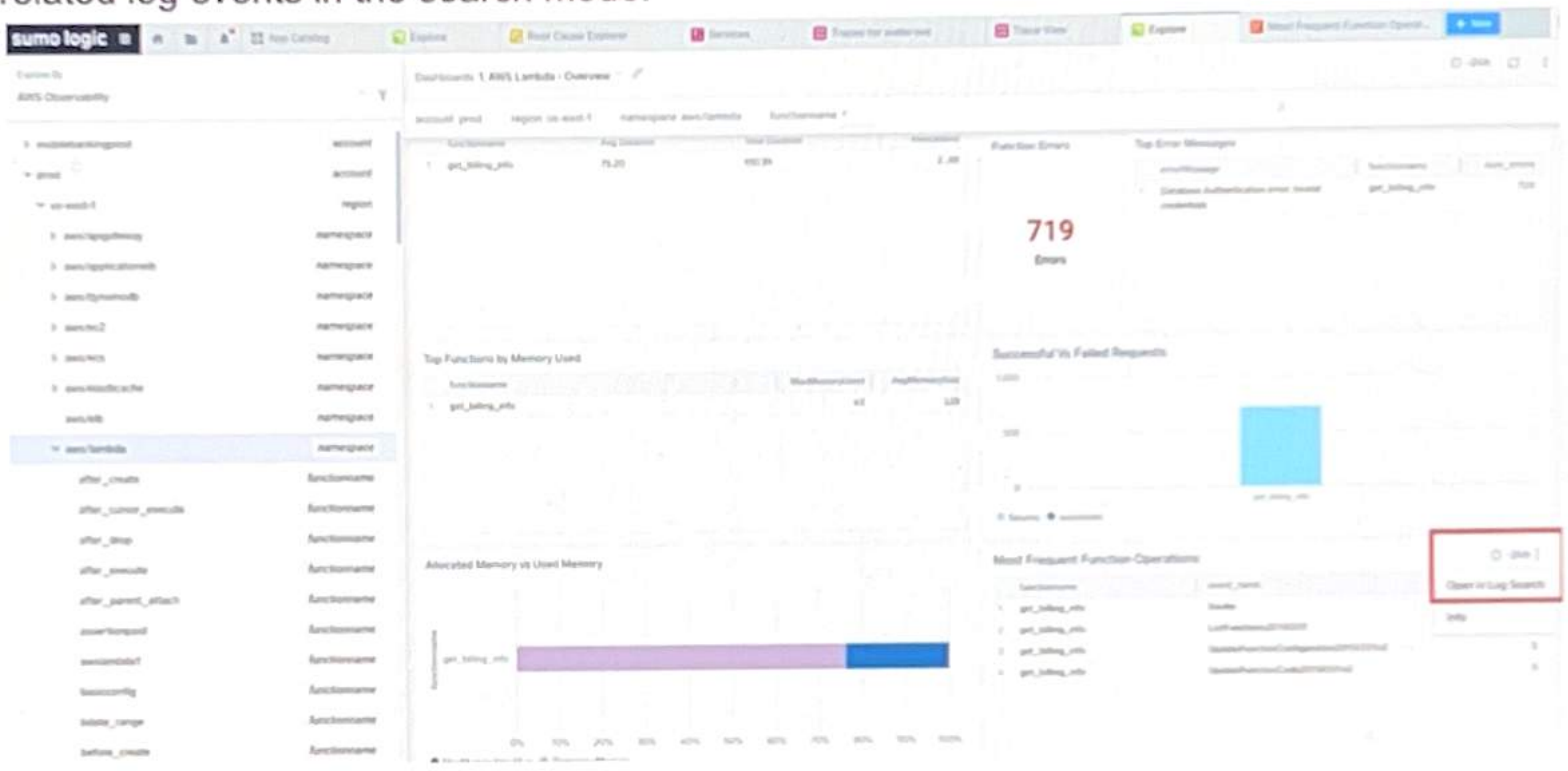


6. In this case, we can see that there were 720 errors related to the ‘get_billing_info’ function just in the last 24 hours. (NOTE: your exact numbers may vary.)

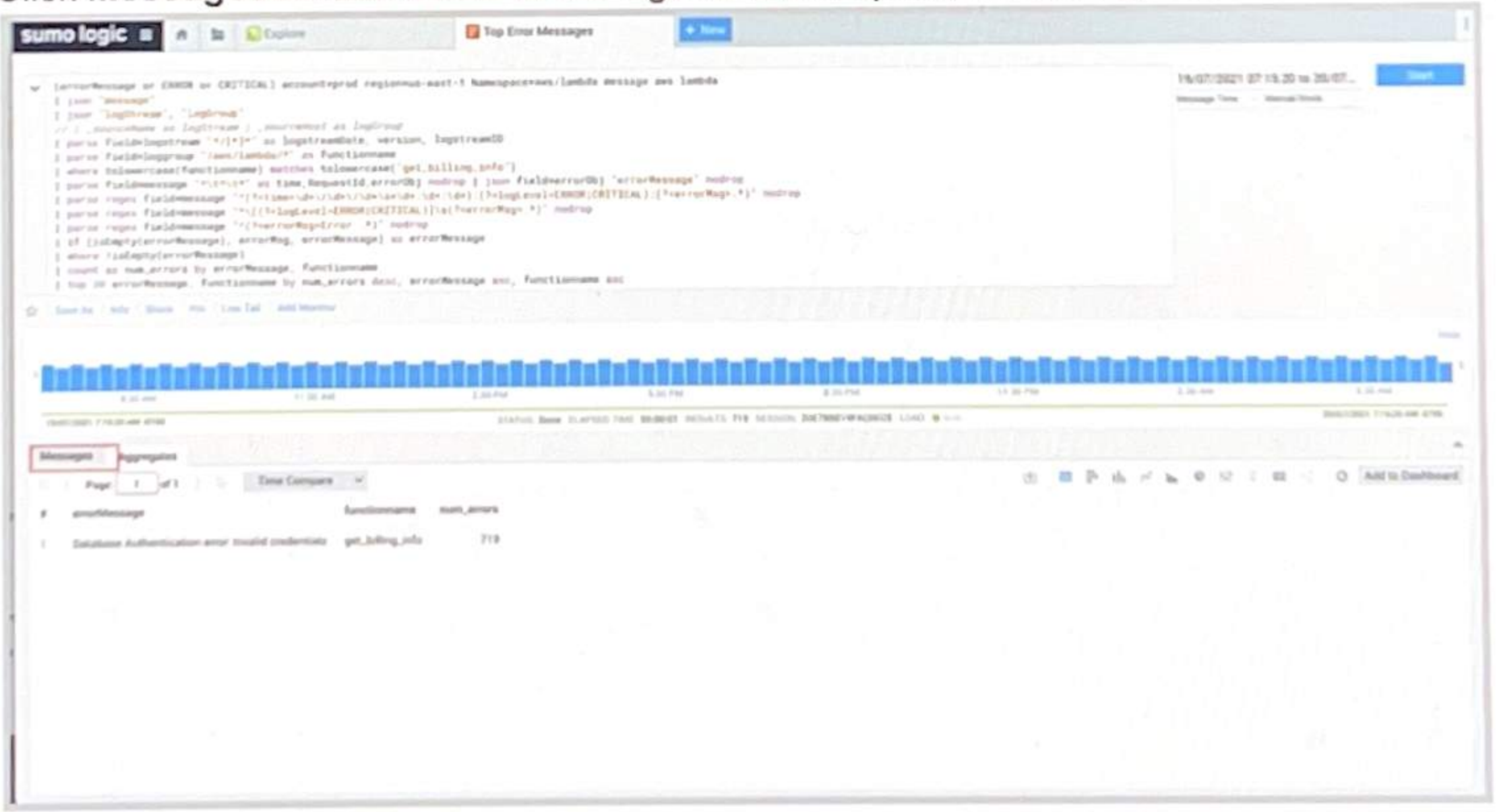
7. Locate the dashboard panel entitled “Successful Vs Failed Requests”. Note here that it appears that 100% of the attempts to call ‘get_billing_info’ failed, indicating

it is likely a fundamental configuration issue, not an occasional communication issue.

8. Find the dashboard directly below called "Most Frequent Function Operations". Note that there are a few calls to "UpdateFunctionConfiguration" that could be the cause of our consistent errors due to misconfiguration. Click the 'triple dot' button on the top right of this panel, then select 'Open in Search' to open the related log events in the search mode.


sumo logic
Dashboards / AWS Lambda - Overview
Function Errors
Top Error Messages
719
Errors
Top Functions by Memory Used
Successful Vs Failed Requests
Allocated Memory vs Used Memory
Most Frequent Function Operations
Open in Log Search

9. Click **Messages** to take a look at the logs for the frequent function operations.

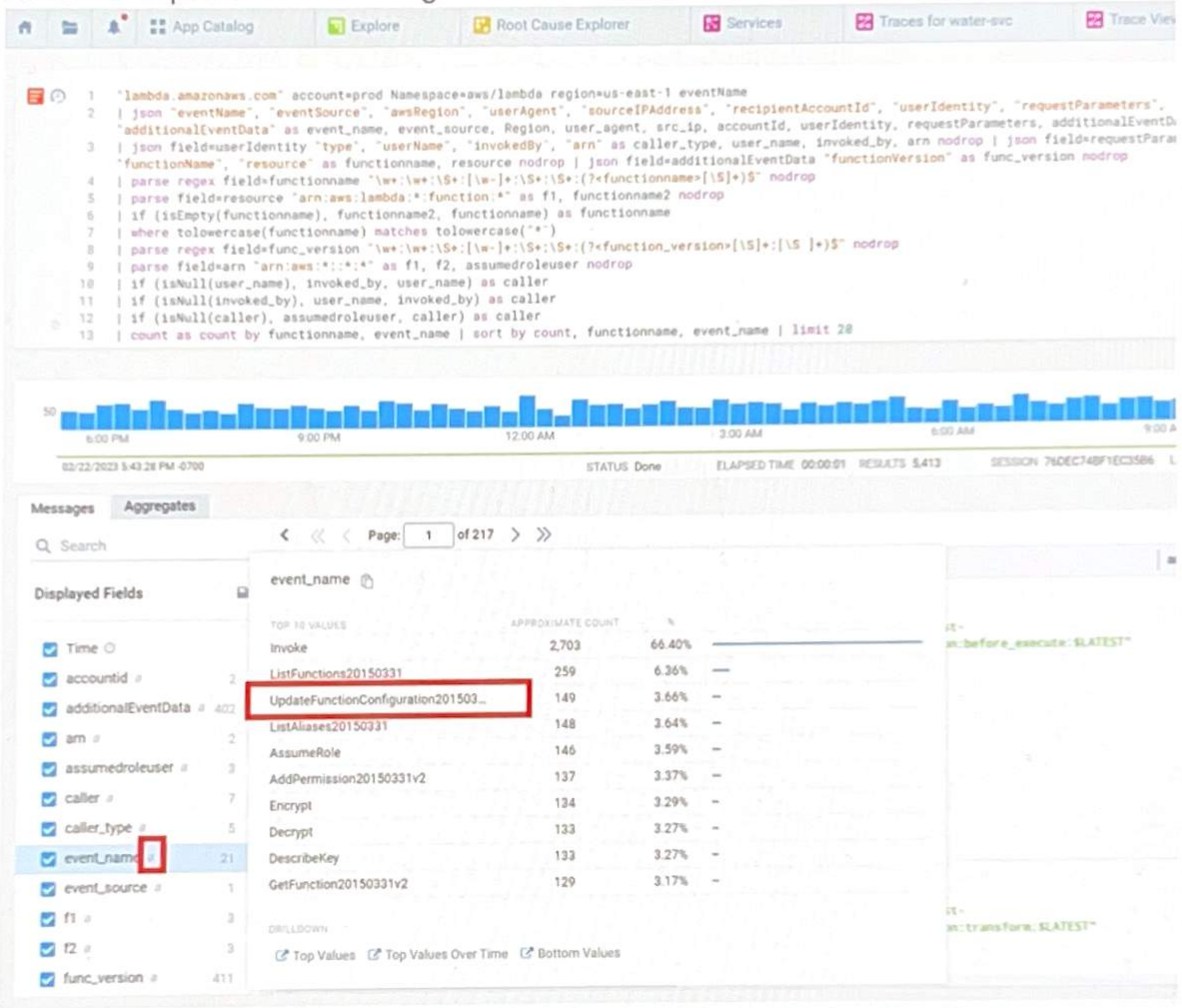


```
(errorMessage or ERROR or CRITICAL) account=prod region=us-east-1 Namespace=aws/lambda message aws lambda
| json "message"
| json "logStream", "logGroup"
// | _sourceName as logStream | _sourceHost as logGroup
| parse field=logstream "*/[*]*" as logstreamDate, version, logstreamID
| parse field=loggroup "/aws/lambda/*" as functionname
| where tolowercase(functionname) matches tolowercase("get_billing_info")
| parse field=message "*\t*\t*" as time,RequestId,errorObj nodrop | json field=errorObj "errorMessage" nodrop
| parse regex field=message [illegible] nodrop
| parse regex field=message [illegible] nodrop
| parse regex field=message "(?<errorMsg>Error: .*)" nodrop
| if (isEmpty(errorMessage), errorMsg, errorMessage) as errorMessage
| where !isEmpty(errorMessage)
| count as num_errors by errorMessage, functionname
| top 10 errorMessage, functionname by num_errors desc, errorMessage asc, functionname asc
```

#	errorMessage	functionname	num_errors
1	Database Authentication error: Invalid credentials	get_billing_info	719

10. Use a filter to view the specific entries for the UpdateFunctionConfiguration

commands. In the **Displayed Fields** list, find the entry for “event_name” then click the small “a” next to the name. On the popup window, select the entry that starts with “UpdateFunctionConfiguration”.



```
"lambda.amazonaws.com" account=prod Namespace=aws/lambda region=us-east-1 eventName
| json "eventName", "eventSource", "awsRegion", "userAgent", "sourceIPAddress", "recipientAccountId", "userIdentity", "requestParameters",
"additionalEventData" as event_name, event_source, Region, user_agent, src_ip, accountId, userIdentity, requestParameters, additionalEventD
| json field=userIdentity "type", "userName", "invokedBy", "arn" as caller_type, user_name, invoked_by, arn nodrop | json field=requestPara
"functionName", "resource" as functionname, resource nodrop | json field=additionalEventData "functionVersion" as func_version nodrop
| parse regex field=functionname "\w+:\w+:\S+:[\w-]+:\S+:\S+:(?<functionname>[\S]+)$" nodrop
| parse field=resource "arn:aws:lambda:*:function:*" as f1, functionname2 nodrop
| if (isEmpty(functionname), functionname2, functionname) as functionname
| where tolowercase(functionname) matches tolowercase("*")
| parse regex field=func_version "\w+:\w+:\S+:[\w-]+:\S+:\S+:(?<function_version>[\S]+:[\S ]+)$" nodrop
| parse field=arn "arn:aws:*::*:*" as f1, f2, assumedroleuser nodrop
| if (isNull(user_name), invoked_by, user_name) as caller
| if (isNull(invoked_by), user_name, invoked_by) as caller
| if (isNull(caller), assumedroleuser, caller) as caller
| count as count by functionname, event_name | sort by count, functionname, event_name | limit 20
```

02/22/2023 5:43:28 PM -0700   STATUS Done   ELAPSED TIME 00:00:01   RESULTS 5,413

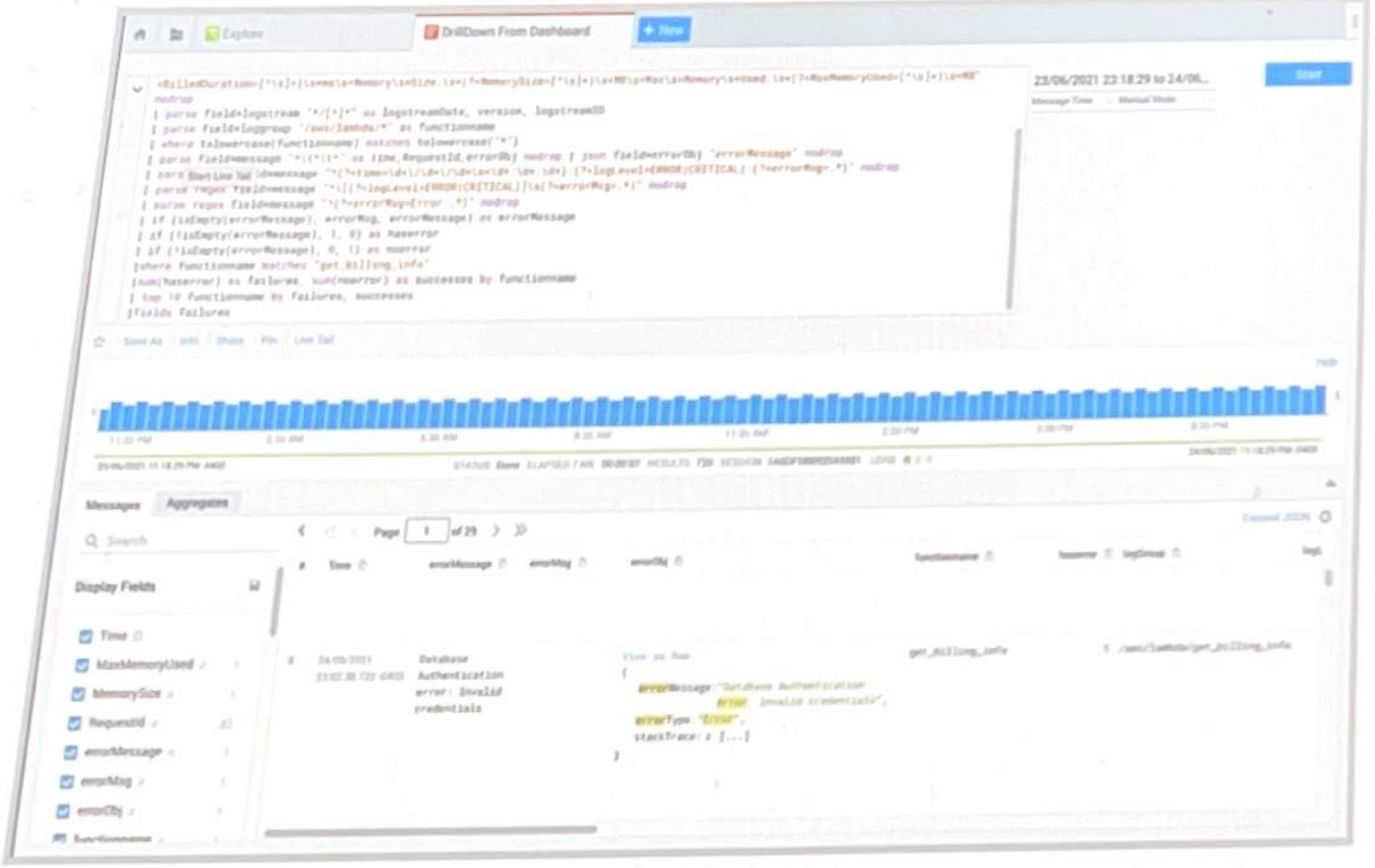
Page: 1 of 217

Time	
accountid	2
additionalEventData	402
arn	2
assumedroleuser	3
caller	7
caller_type	5
event_name	21
event_source	1
f1	3
f2	3
func_version	411

event_name

TOP 10 VALUES	APPROXIMATE COUNT	%
Invoke	2,703	66.40%
ListFunctions20150331	259	6.36%
UpdateFunctionConfiguration201503...	149	3.66%
ListAliases20150331	148	3.64%
AssumeRole	146	3.59%
AddPermission20150331v2	137	3.37%
Encrypt	134	3.29%
Decrypt	133	3.27%
DescribeKey	133	3.27%
GetFunction20150331v2	129	3.17%

11. Examine the resulting logs to find that one of the developers has been making the changes in the function configuration that likely caused the constant failures. You’ll see in the filtered log messages that it was **Mark Smith** who was trying to change the configuration (click on the “a” next to the “user_name” to see the recorded user names for these log entries).. We can now contact Mark to find out the reason why he was changing the details, and rectify the issue.



```
<BilledDuration=[^\s]+)\s+ms\s+Memory\s+Size:\s+(?<MemorySize=[^\s]+)\s+MB\s+Max\s+Memory\s+Used:\s+(?<MaxMemoryUsed>[^\s]+)\s+MB"
nodrop
| parse field=logstream "*/[*]*" as logstreamDate, version, logstreamID
| parse field=loggroup "/aws/lambda/*" as functionname
| where tolowercase(functionname) matches tolowercase("*")
| parse field=message "*:\t*\t*" as time,RequestId,errorObj nodrop | json field=errorObj "errorMessage" nodrop
| parse regex field=message "^(?<time>\d+\/\d+\/\d+\s+\d+:\d+:\d+) (?<logLevel>ERROR|CRITICAL) (?<errorMsg>.*)" nodrop
| parse regex field=message "^\[(?<logLevel>ERROR|CRITICAL)]\s(?<errorMsg>.*)" nodrop
| parse regex field=message "^(?<errorMsg>Error .*)" nodrop
| if (isEmpty(errorMessage), errorMsg, errorMessage) as errorMessage
| if (!isEmpty(errorMessage), 1, 0) as haserror
| if (!isEmpty(errorMessage), 0, 1) as noerror
|where functionname matches "get_billing_info"
|sum(haserror) as failures, sum(noerror) as successes by functionname
| top 10 functionname by failures, successes
|fields failures
```

```
#  Time                 errorMessage               errorObj
1  24/06/2021           Database                   View as Raw
   23:02:38.723 -0400   Authentication             {
                        error: Invalid               errorMessage:"Database Authentication
                        credentials                              error: Invalid credentials",
                                                     errorType:"Error",
                                                     stackTrace: # [...]
                                                   }
functionname: get_billing_info
logGroup: /aws/lambda/get_billing_info
```

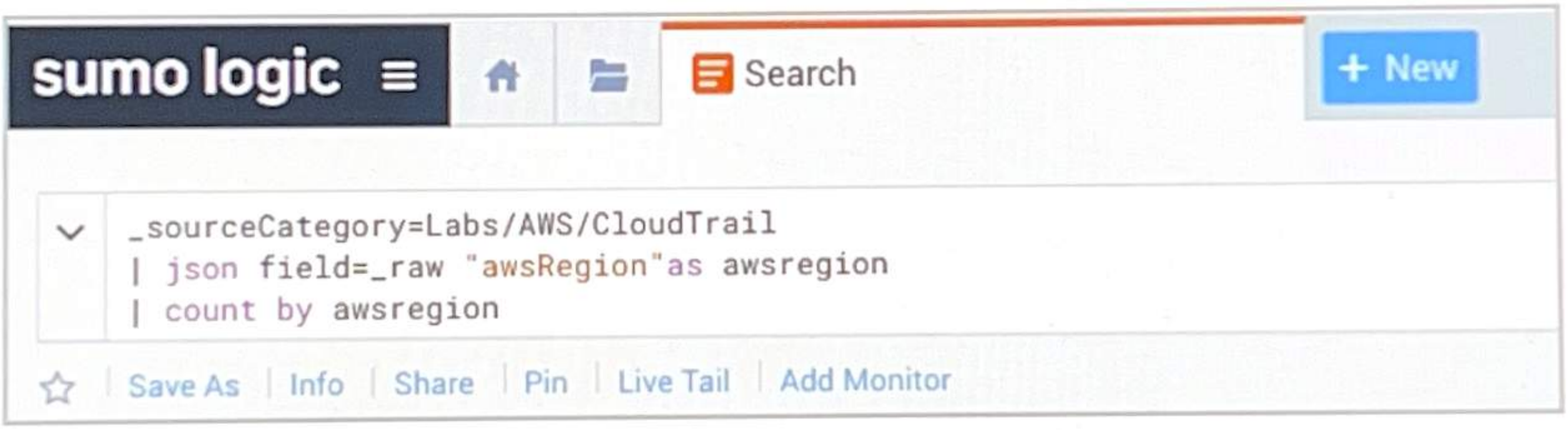
Lab 3: Working with Logs

In this lab, you'll learn how to use some additional tools to filter and analyze logs, as well as explore the LogReduce® and LogCompare operators.

Note: All labs assume you're using a training+analyst###@sumologic.com account. The data you see may vary depending on your environment if you're using your own credentials or a Sumo Logic trial account instead of a training account. To access a training account, review **Lab 0**.

The lab begins.

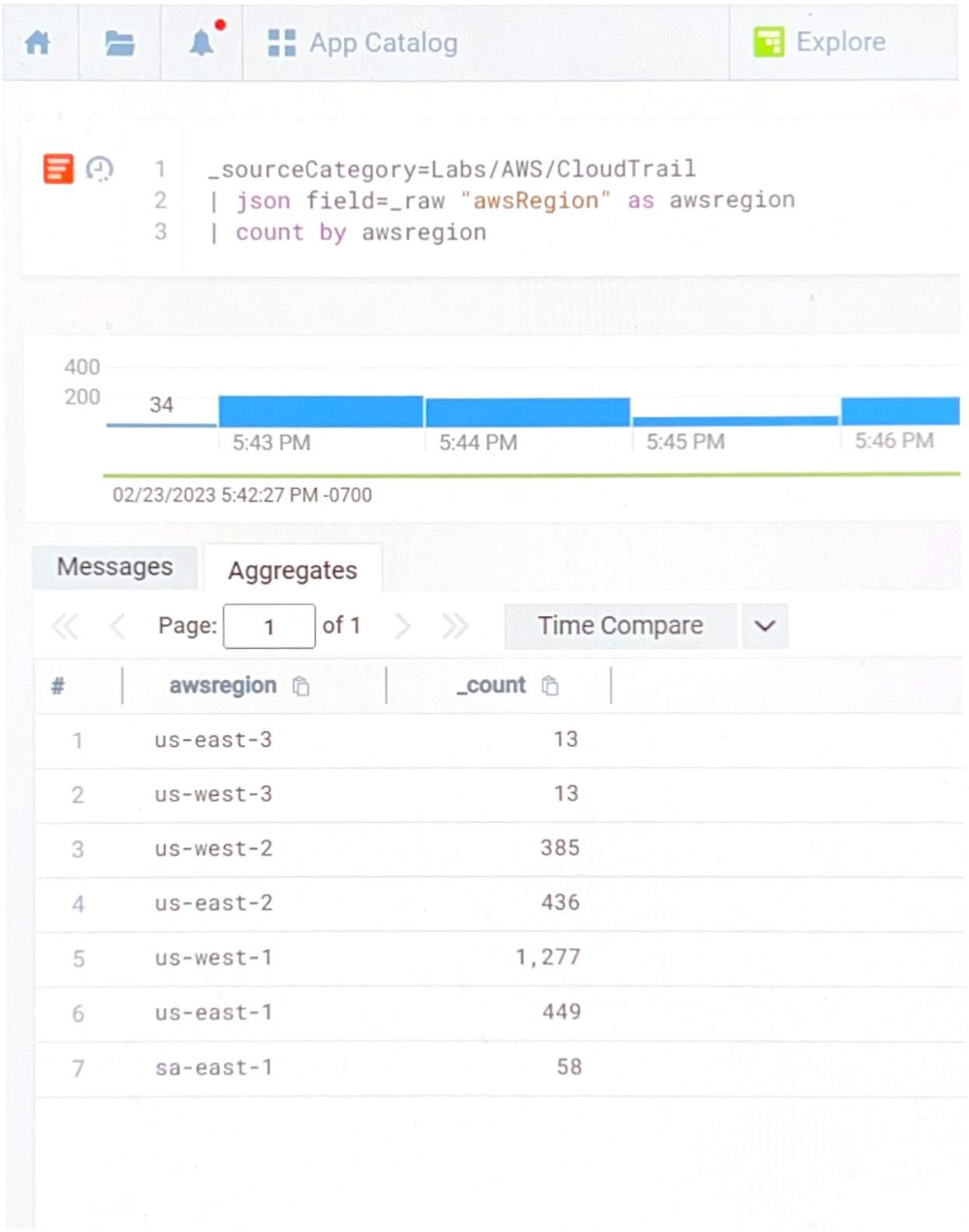
1. In the top right, click **+New**, then **Log Search**.
2. In the search query box, type or copy/paste the following query:
```
_sourceCategory=Labs/AWS/CloudTrail
| json field=_raw "awsRegion"as awsregion
| count by awsregion
```



```
_sourceCategory=Labs/AWS/CloudTrail
| json field=_raw "awsRegion"as awsregion
| count by awsregion
```

3. Click **Start** to execute the query.

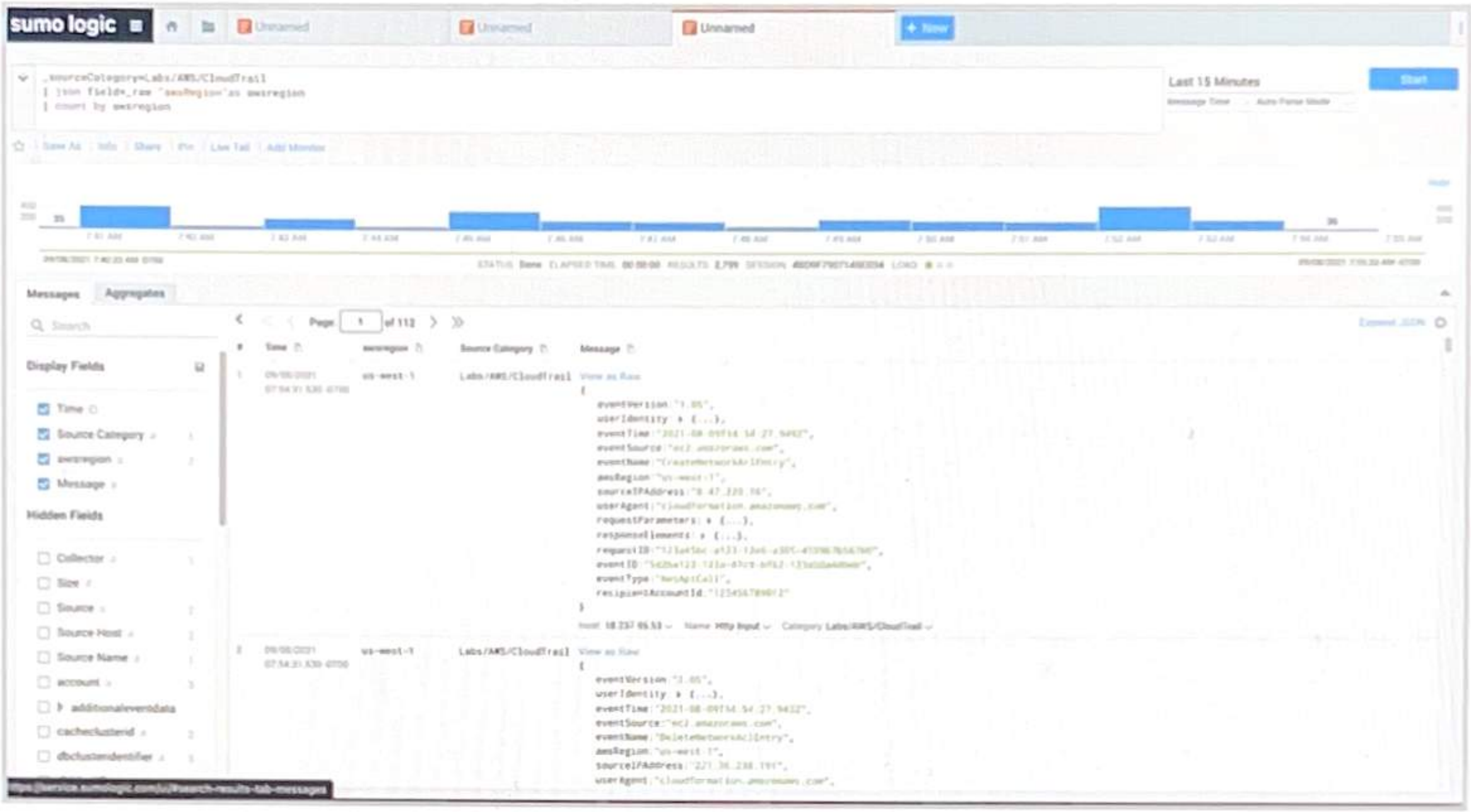
   The **Aggregate** panel shows the count by region.



```
_sourceCategory=Labs/AWS/CloudTrail
| json field=_raw "awsRegion" as awsregion
| count by awsregion
```

#	awsregion	_count
1	us-east-3	13
2	us-west-3	13
3	us-west-2	385
4	us-east-2	436
5	us-west-1	1,277
6	us-east-1	449
7	sa-east-1	58

4. Click **Messages**. The individual Log entries will be displayed. Use the page navigation arrows to move within the log pages.



```
_sourceCategory=Labs/AWS/CloudTrail
| json field=_raw "awsRegion" as awsregion
| count by awsregion
```

```
#  Time                          awsregion   Source Category       Message
1  09/08/2021 07:54:31.530 -0700 us-west-1   Labs/AWS/CloudTrail   View as Raw
                                                                   {
                                                                     eventVersion:"1.05",
                                                                     userIdentity: ▸ {...},
                                                                     eventTime:"2021-08-09T14:54:27.949Z",
                                                                     eventSource:"ec2.amazonaws.com",
                                                                     eventName:"CreateNetworkAclEntry",
                                                                     awsRegion:"us-west-1",
                                                                     sourceIPAddress:"8.47.220.76",
                                                                     userAgent:"cloudformation.amazonaws.com",
                                                                     requestParameters: ▸ {...},
                                                                     responseElements: ▸ {...},
                                                                     requestID:"[illegible]",
                                                                     eventID:"[illegible]",
                                                                     eventType:"AwsApiCall",
                                                                     recipientAccountId:"123456789012"
                                                                   }
                                                                   Host: 18.237.65.53 ⌄  Name: Http Input ⌄  Category: Labs/AWS/CloudTrail ⌄
2  09/08/2021 07:54:31.530 -0700 us-west-1   Labs/AWS/CloudTrail   View as Raw
                                                                   {
                                                                     eventVersion:"1.05",
                                                                     userIdentity: ▸ {...},
                                                                     eventTime:"2021-08-09T14:54:27.943Z",
                                                                     eventSource:"ec2.amazonaws.com",
                                                                     eventName:"DeleteNetworkAclEntry",
                                                                     awsRegion:"us-west-1",
                                                                     sourceIPAddress:"221.36.238.191",
                                                                     userAgent:"cloudformation.amazonaws.com",
```

Use the **field browser** to filter the log entries as per your requirement.

LogReduce® Operator:

The LogReduce® algorithm uses fuzzy logic to group messages together based on string and pattern similarity. You can use the "logreduce" operator in queries to quickly assess activity patterns for things like a range of devices or traffic on a website. Focus the LogReduce® algorithm on an area of interest by defining that area in the keyword expression.

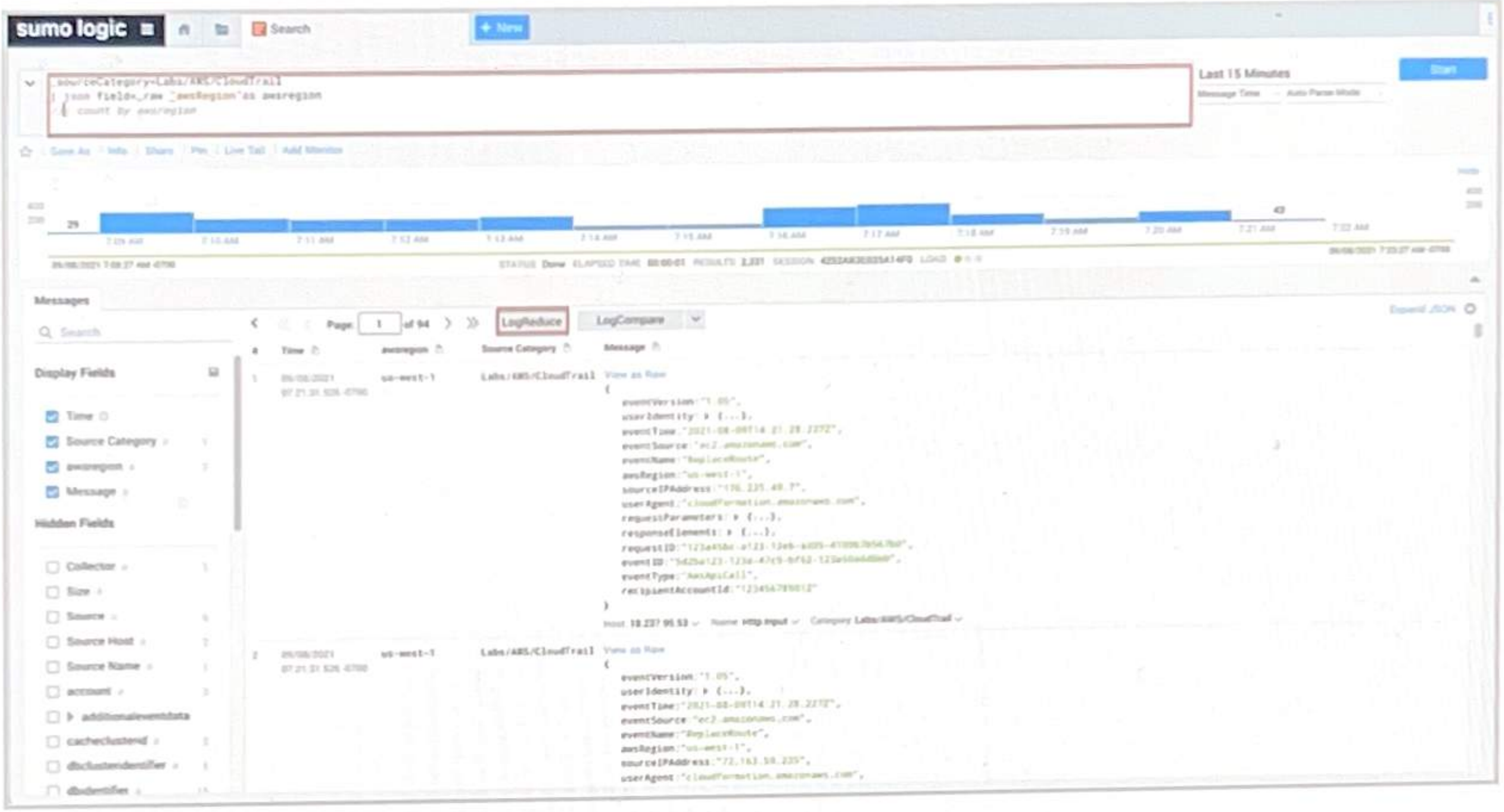
There are two ways to use the operator.

- Use the **LogReduce** button displayed on the results table after running a search.
- Manually add the "logreduce" operator to your query following its syntax.

For our lab, we will use the LogReduce button.

1. First, modify our query to remove the 'count by' clause – the "logreduce" operator cannot be used with aggregation or group-by operators such as "count by field".

```
_sourceCategory=Labs/AWS/CloudTrail
| json field=_raw "awsRegion"as awsregion
```



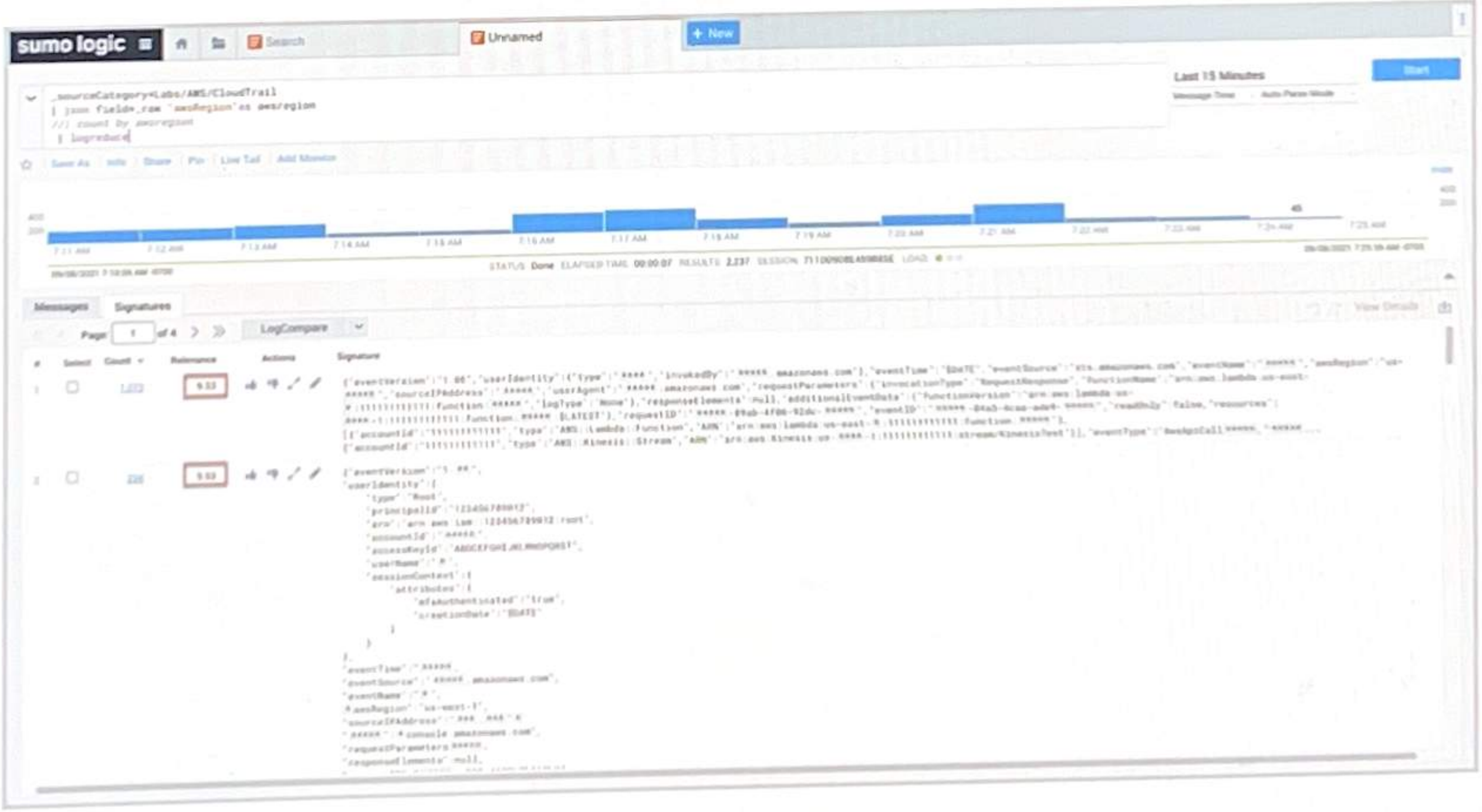
```
_sourceCategory=Labs/AWS/CloudTrail
| json field=_raw "awsRegion" as awsregion
| count by awsregion
```

#	Time	awsregion	Source Category	Message
1	09/08/2021 07:21:31.526 -0700	us-west-1	Labs/AWS/CloudTrail	View as Raw
2	09/08/2021 07:21:31.526 -0700	us-west-1	Labs/AWS/CloudTrail	View as Raw

```
{
    eventVersion:"1.05",
    userIdentity: ▸ {...},
    eventTime:"2021-08-09T14:21:28.227Z",
    eventSource:"ec2.amazonaws.com",
    eventName:"ReplaceRoute",
    awsRegion:"us-west-1",
    sourceIPAddress:"176.225.48.7",
    userAgent:"cloudformation.amazonaws.com",
    requestParameters: ▸ {...},
    responseElements: ▸ {...},
    requestID:"123e456e-a123-12eb-a005-4100b7b56[illegible]",
    eventID:"5d2ba123-123d-47c9-bf62-123e[illegible]",
    eventType:"AwsApiCall",
    recipientAccountId:"123456789012"
}
```

2. Click the **LogReduce** button.

3. The **Signatures** tab is displayed with the results. Note the "Relevance" column which indicates the category of log messages that are judged to be most relevant to the query.



```
_sourceCategory=Labs/AWS/CloudTrail
| json field=_raw "awsRegion" as awsregion
//| count by awsregion
| logreduce
```

#	Select	Count	Relevance	Signature
1	☐	1,223	9.53	{'eventVersion':'1.##','userIdentity':{'type':'####','invokedBy':'#####.amazonaws.com'},'eventTime':'$DATE','eventSource':'#####.amazonaws.com','eventName':'#####','awsRegion':'us-[illegible]
2	☐	226	9.53	{'eventVersion':'1.##', 'userIdentity':{ 'type':'Root', 'principalId':'123456789012', 'arn':'arn:aws:iam::123456789012:root', 'accountId':'#####', 'accessKeyId':'ABCDEFGHIJKLMNOPQRST', 'userName':'#', 'sessionContext':{ 'attributes':{ 'mfaAuthenticated':'true', 'creationDate':'$DATE' } } }, 'eventTime':'#####', 'eventSource':'#####.amazonaws.com', 'eventName':'#', 'awsRegion':'us-west-1', 'sourceIPAddress':'###.###.#', '#####':'#console.amazonaws.com', 'requestParameters':#####, 'responseElements':null,

We can further explore the functionality of LogReduce® which allows you to distill unique messages from the noise by identifying recurring Signatures in your data.

4. Search for all messages that have the word "error" across all your data, use:

```
error | logreduce
```

5. Then click on **LogReduce** to get a summarized view of all messages.

6. Next, in a new search, run LogReduce® on your Snort security data to identify unusual activity (i.e. intrusions).

```
_sourceCategory=labs/snort

| logreduce
```

7. Sort your results by count to identify those that happen only once.

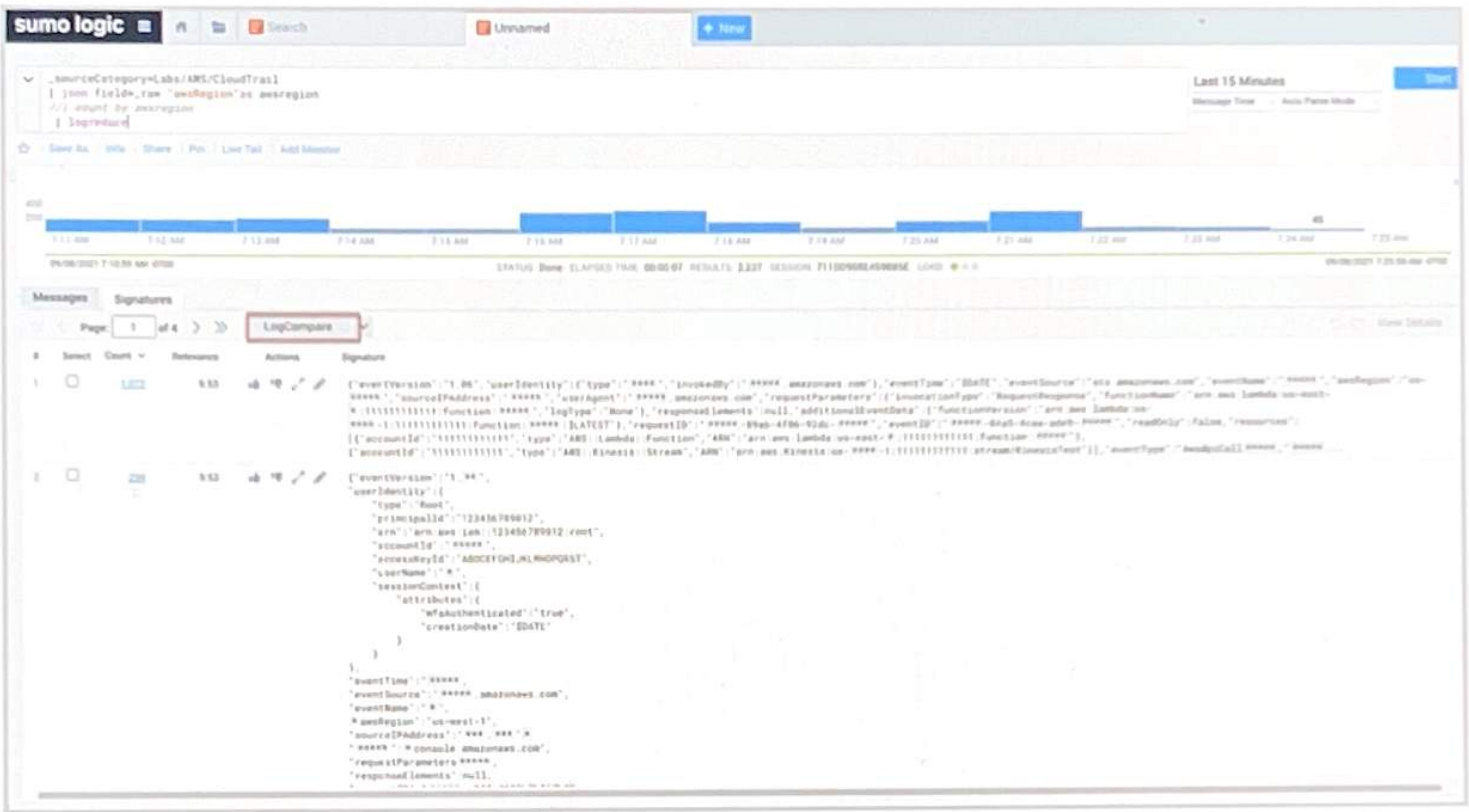
8. Click on the count (1) to view the unusual message.

9. Now click on the host to view surrounding messages to identify the context of the intrusion.

"logcompare" Operator:
The "logcompare" operator allows you to compare log activity from two different time periods, providing you insight on how your current time compares to a baseline. In this case, we will use the logcompare operator to identify when signature messages deviate by more than 25% from the baseline.

1. Use logcompare to run a summarized query for a baseline 24 hours ago.

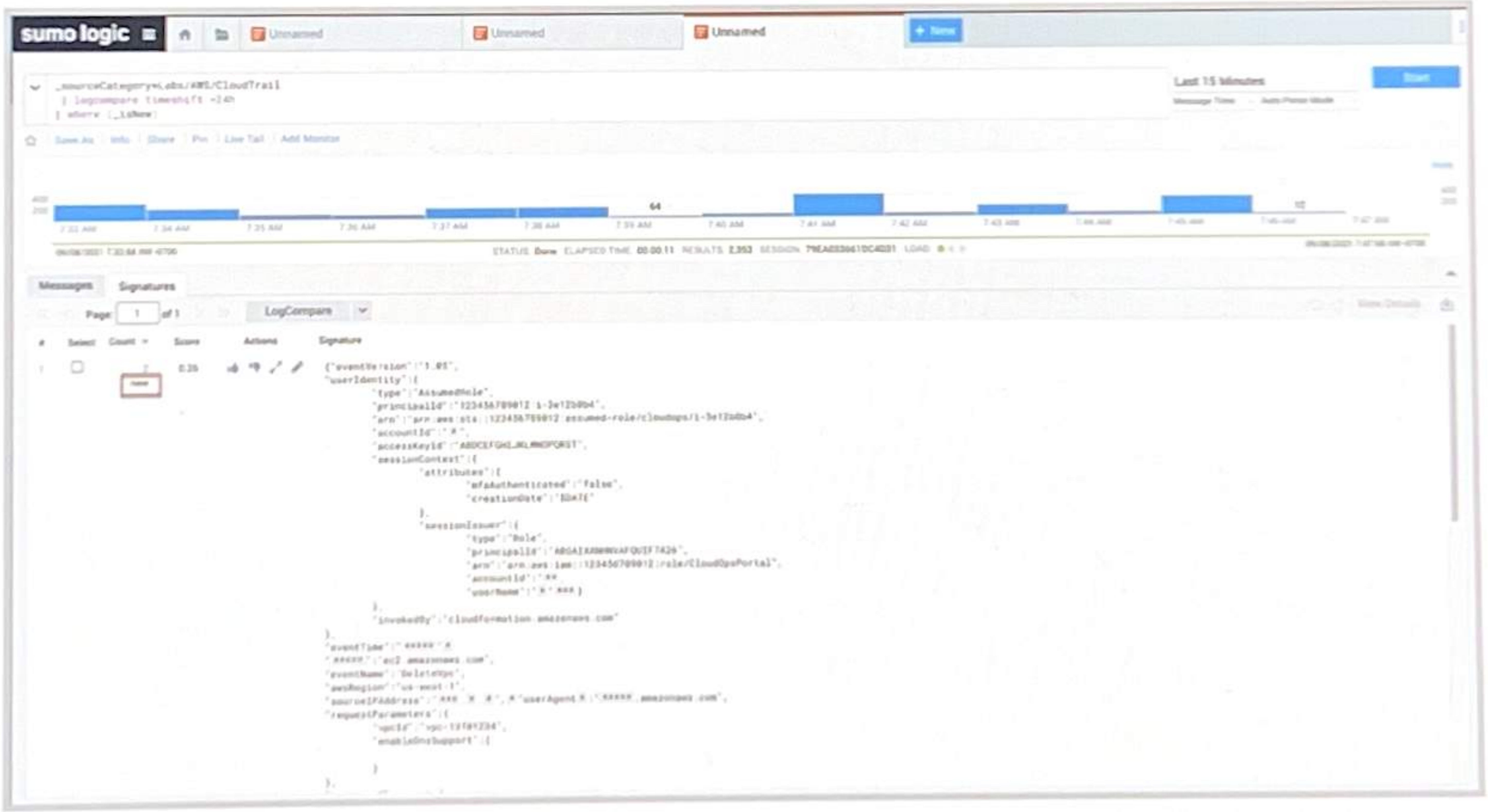
```
_sourceCategory=Labs/AWS/CloudTrail
 | logcompare timeshift -24h
```



```
_sourceCategory=Labs/AWS/CloudTrail
| json field=_raw "awsRegion" as awsregion
//| count by awsregion
 | logreduce
```

```
{"eventVersion":"1.06","userIdentity":{"type":"****","invokedBy":"***** amazonaws.com"},"eventTime":"$DATE","eventSource":"sts.amazonaws.com","eventName":"*****","awsRegion":"us-
*****","sourceIPAddress":"*****","userAgent":"***** amazonaws.com","requestParameters":{"invocationType":"RequestResponse","functionName":"arn:aws:lambda:us-east-
*:111111111111:function:*****","logType":"None"},"responseElements":null,"additionalEventData":{"functionVersion":"arn:aws:lambda:us-
****-1:111111111111:function:*****:$LATEST"},"requestID":"*****-89ab-4f86-92dc-*****","eventID":"*****-8da5-4cea-adeb-*****","readOnly":false,"resources":
[{"accountId":"111111111111","type":"AWS::Lambda::Function","ARN":"arn:aws:lambda:us-east-*:111111111111:function:*****"},
{"accountId":"111111111111","type":"AWS::Kinesis::Stream","ARN":"arn:aws:Kinesis:us-****-1:111111111111:stream/KinesisTest"}],"eventType":"AwsApiCall",***** ,*****
```

```
{"eventVersion":"1.**",
"userIdentity":{
    "type":"Root",
    "principalId":"123456789012",
    "arn":"arn:aws:iam::123456789012:root",
    "accountId":"*****",
    "accessKeyId":"ABDCEFGHIJKLMNOPQRST",
    "userName":"*",
    "sessionContext":{
        "attributes":{
            "mfaAuthenticated":"true",
            "creationDate":"$DATE"
        }
    }
},
"eventTime":"*****",
"eventSource":"***** amazonaws.com",
"eventName":"*",
*awsRegion":"us-west-1",
"sourceIPAddress":"***.***.*
"*****":*console.amazonaws.com",
"requestParameters*****,
"responseElements":null,
```



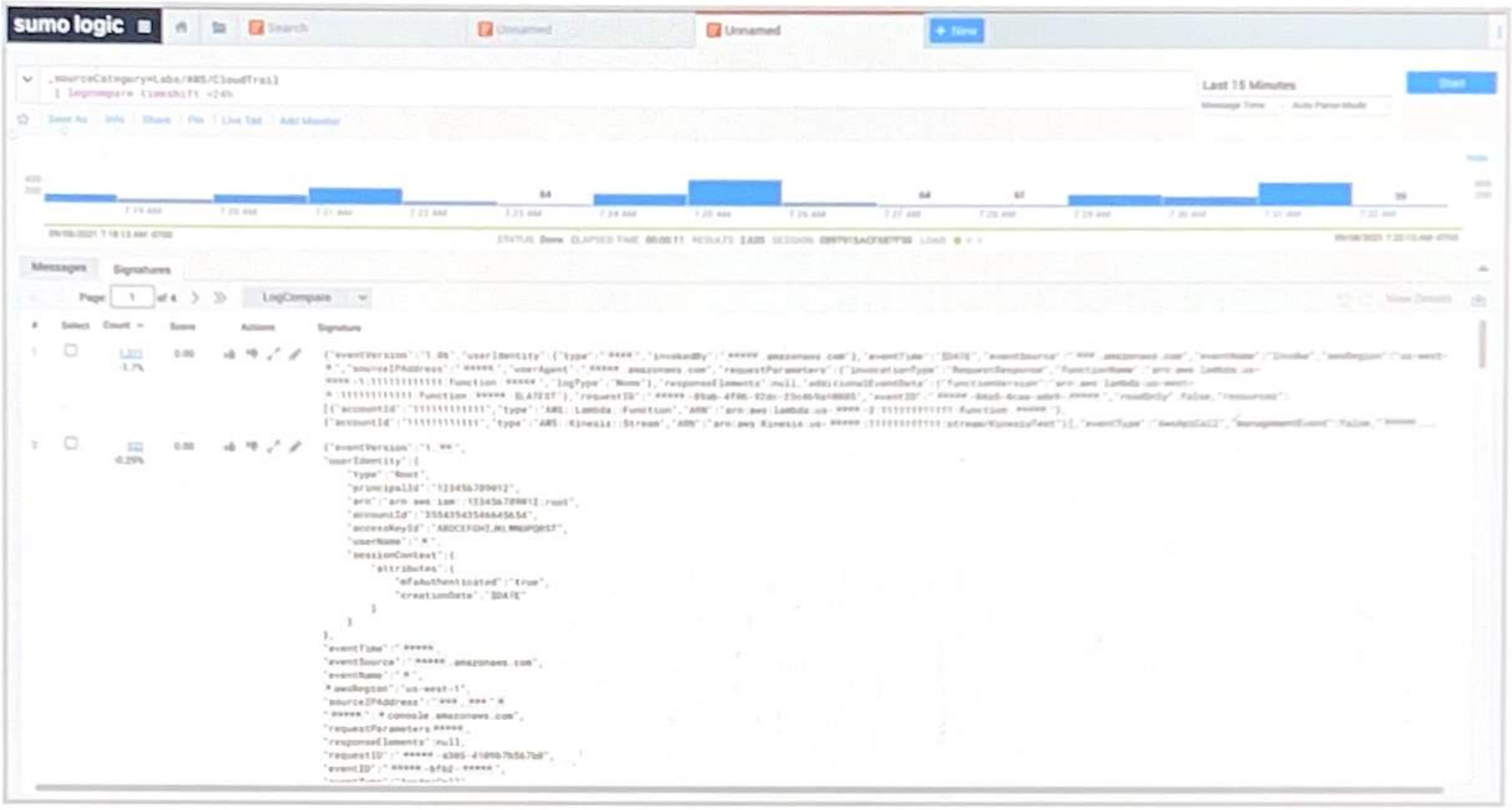
```
_sourceCategory=Labs/AWS/CloudTrail
 | logcompare timeshift -24h
| where (_isNew)
```

```
{"eventVersion":"1.05",
"userIdentity":{
        "type":"AssumedRole",
        "principalId":"123456789012:i-3e123db4",
        "arn":"arn:aws:sts::123456789012:assumed-role/cloudops/i-3e123db4",
        "accountId":"*",
        "accessKeyId":"ABDCEFGHIJKLMNOPQRST",
        "sessionContext":{
                "attributes":{
                        "mfaAuthenticated":"false",
                        "creationDate":"$DATE"
                },
                "sessionIssuer":{
                        "type":"Role",
                        "principalId":"AROAIXXXXXXAFQUIF7A26",
                        "arn":"arn:aws:iam::123456789012:role/CloudOpsPortal",
                        "accountId":"**
                        "userName":"*"***}
        },
        "invokedBy":"cloudformation.amazonaws.com"
},
"eventTime":"*****"*
"*****":"ec2.amazonaws.com",
"eventName":"DeleteVpc",
"awsRegion":"us-west-1",
"sourceIPAddress":"***.*.*",*"userAgent*:"*****.amazonaws.com",
"requestParameters":{
        "vpcId":"vpc-1f1f1234",
        "enableDnsSupport":{

        }
},
```

2. Note that in some cases the delta (percentage change) is very small and not significant. To view only those results where the delta percentage is significant (let's say: greater than 25%), let's add a 'where' clause for _deltaPercentage.

```
_sourceCategory=Labs/AWS/CloudTrail
| logcompare timeshift -24h
| where abs(_deltaPercentage) > 25
```

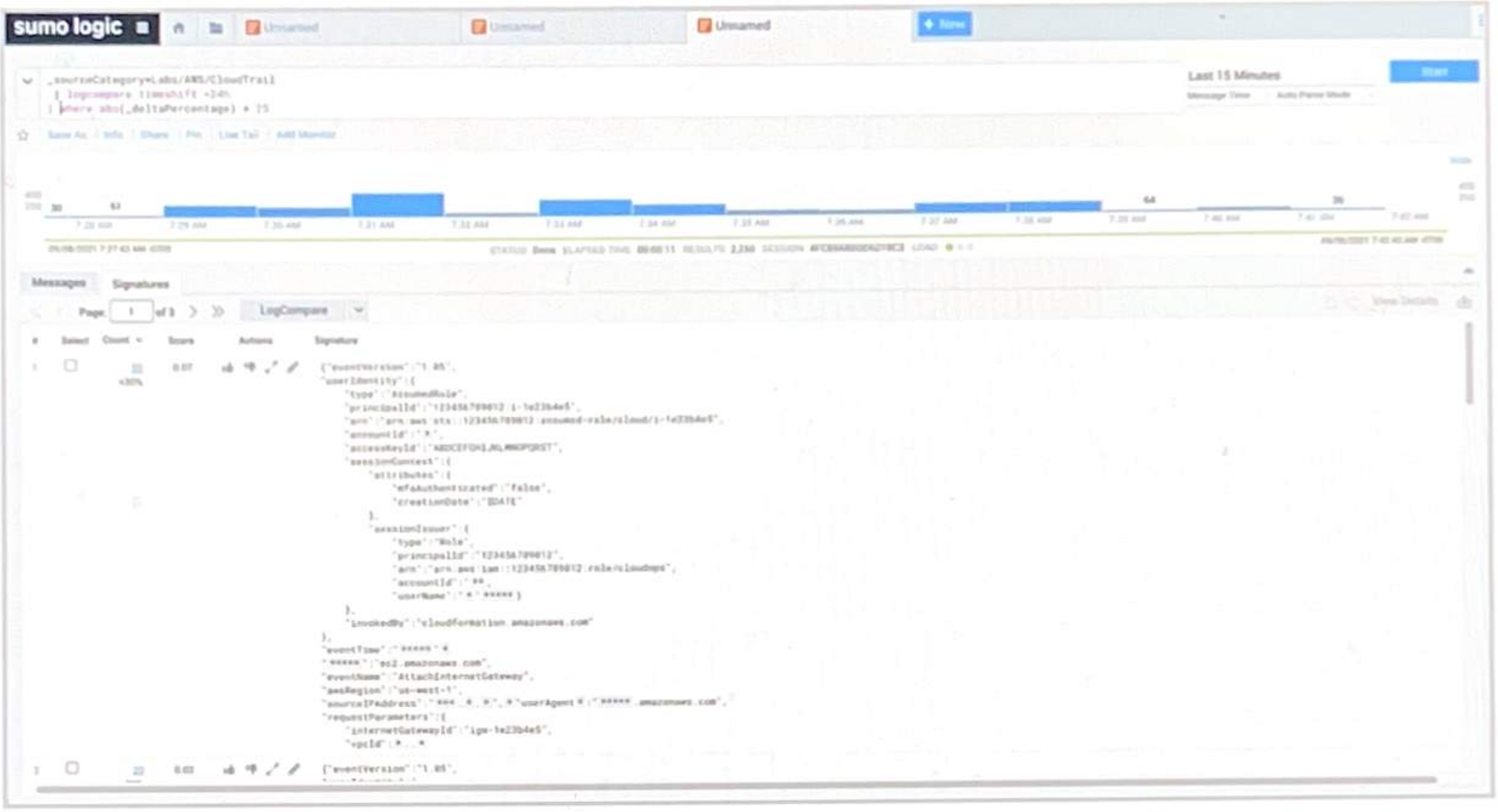


```
_sourceCategory=Labs/AWS/CloudTrail
| logcompare timeshift -24h
```

#	Select	Count	Score	Actions	Signature
1	☐	1,321 -1.7%	0.00		{"eventVersion":"1.08","userIdentity":{"type":"****","invokedBy":"***** amazonaws.com"},"eventTime":"$DATE","eventSource":"*** amazonaws.com","eventName":"Invoke","awsRegion":"us-west-*","sourceIPAddress":"*****","userAgent":"***** amazonaws.com","requestParameters":{"invocationType":"RequestResponse","functionName":"arn:aws:lambda:us-****-1:111111111111:function:*****","logType":"None"},"responseElements":null,"additionalEventData":{"functionVersion":"arn:aws:lambda:us-west-*:111111111111:function:*****:$LATEST"},"requestID":"*****-8bab-4f06-92dc-[illegible]","eventID":"*****-[illegible]-*****","readOnly":false,"resources":[{"accountId":"111111111111","type":"AWS::Lambda::Function","ARN":"arn:aws:lambda:us-****-2:111111111111:function:*****"},{"accountId":"111111111111","type":"AWS::Kinesis::Stream","ARN":"arn:aws:kinesis:us-*****:111111111111:stream/KinesisTest"}],"eventType":"AwsApiCall","managementEvent":false,"*****...
2	☐	332 -0.29%	0.00		{"eventVersion":"1.**", "userIdentity":{ "type":"Root", "principalId":"123456789012", "arn":"arn:aws:iam::123456789012:root", "accountId":"355435435466445654", "accessKeyId":"ABDCEFGHIJKLMNOPQRST", "userName":"*", "sessionContext":{ "attributes":{ "mfaAuthenticated":"true", "creationDate":"$DATE" } } }, "eventTime":"*****, "eventSource":"***** amazonaws.com", "eventName":"*", "awsRegion":"us-west-1", "sourceIPAddress":"***.***"*, "*****":"*console.amazonaws.com", "requestParameters *****, "responseElements":null, "requestID":"*****-a385-410f[illegible]", "eventID":"*****-bf62-*****",

3. We can also look for entirely new messages that did not appear at all during the previous baseline time period. To view results for new Signature messages in the current time period, add a where clause for _isNew:

```
_sourceCategory=Labs/AWS/CloudTrail
| logcompare timeshift -24h
| where (_isNew)
```



```
_sourceCategory=Labs/AWS/CloudTrail
 | logcompare timeshift -24h
| where abs(_deltaPercentage) > 15
```

```
{"eventVersion":"1.05",
"userIdentity":{
    "type":"AssumedRole",
    "principalId":"123456789012:i-1e23b4e5",
    "arn":"arn:aws:sts::123456789012:assumed-role/cloud/i-1e23b4e5",
    "accountId":"*",
    "accessKeyId":"ABDCEFGHIJKLMNOPQRST",
    "sessionContext":{
        "attributes":{
            "mfaAuthenticated":"false",
            "creationDate":"$DATE"
        },
        "sessionIssuer":{
            "type":"Role",
            "principalId":"123456789012",
            "arn":"arn:aws:iam::123456789012:role/cloudops",
            "accountId":"**,
            "userName":"*"*****}
    },
    "invokedBy":"cloudformation.amazonaws.com"
},
"eventTime":"*****"*
"*****":"ec2.amazonaws.com",
"eventName":"AttachInternetGateway",
"awsRegion":"us-west-1",
"sourceIPAddress":"***.*.*",*"userAgent*":"*****.amazonaws.com",
"requestParameters":{
    "internetGatewayId":"igw-1e23b4e5",
    "vpcId":*...*
{"eventVersion":"1.05",
```

LogExplain Operator

The **LogExplain** operator allows you to compare sets of structured logs based on events you're interested in. Structured logs can be in JSON, CSV, key-value, or any structured format.

You'll need to specify an event of interest as a conditional statement – this is called the **Event Condition**. You can specify a condition to compare against the event-of-interest condition, this is called the **Against Condition**. If no Against Condition is provided, LogExplain will generate the comparison data set based on the fields in your Event Condition.

The syntax is:

```
| logexplain <event_condition> [against <against_condition>] on
<fieldname>
```

LogExplain will process your data against the specified conditions and create separate data sets to compare:

- A control data set from normal operations data.
- An event-of-interest data set.

LogExplain gathers frequent (at least 5% higher) joint-column entries, such as key-value pairs that occur more frequently compared to the control set. The results indicate what entities correlate with the event you're interested in.

To explore the LogExplain operator, copy and run the following query on CloudTrail: