Mastering Observability of your Serverless Applications

Chris Munns
Senior Manager/Principal Developer Advocate - Serverless
Amazon Web Services

aws

About me

Chris Munns - munns@amazon.com, @chrismunns

- Sr Manager/Principal Developer Advocate – Serverless
- New Yorker (ehhh…ish.. kids/burbs/ya know?)

  Previously:
  - AWS Business Development Manager – DevOps, July ‘15 - Feb ‘17
  - AWS Solutions Architect Nov ‘11- Dec ‘14
  - Formerly on operations teams @Etsy and @Meetup
  - Little time at a hedge fund, Xerox and a few other startups
- Rochester Institute of Technology: Applied Networking and Systems Administration ’05
- Internet infrastructure geek

aws

https://secure.flickr.com/photos/mgifford/4525333972

Serverless applications

Event source

Changes in data state

Requests to endpoints

Changes in resource state

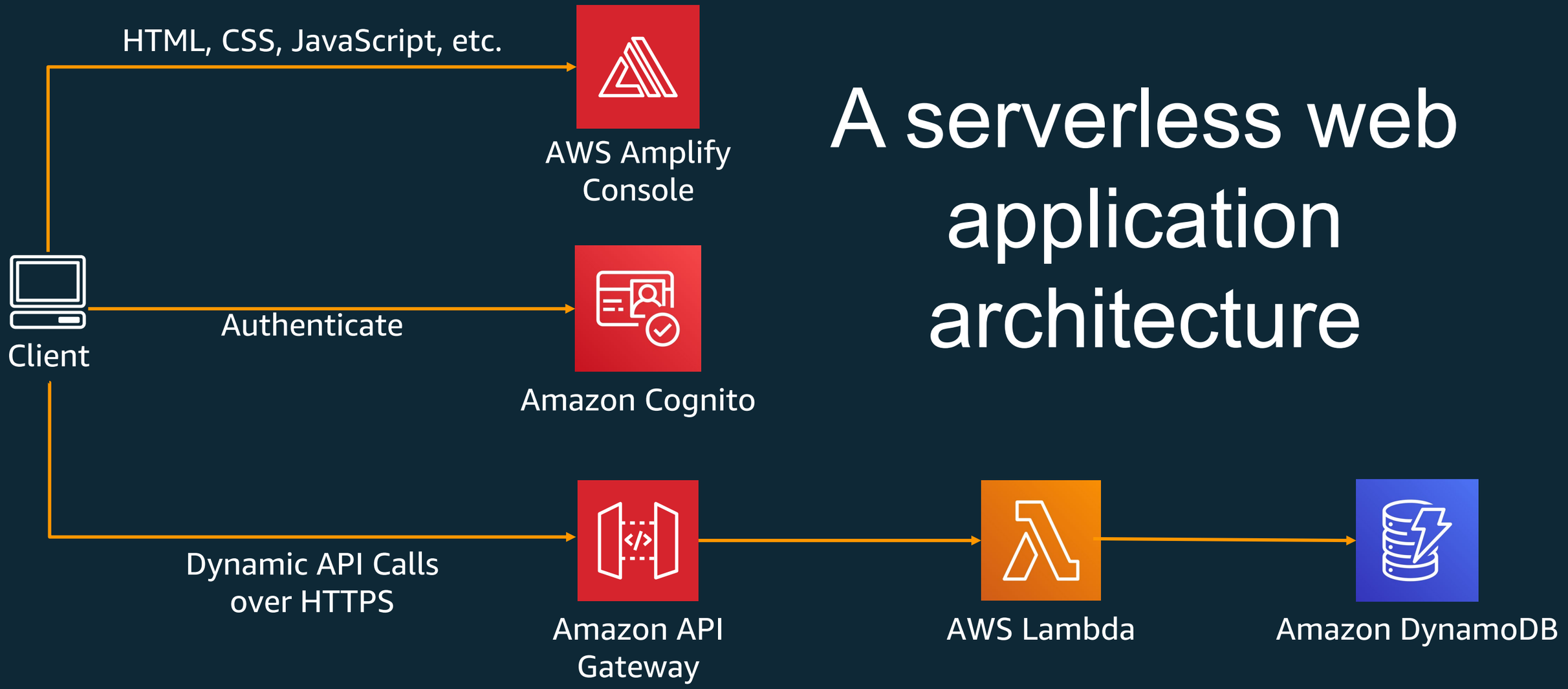
Function

Node.js
Python
Java
C#
Go
Ruby
Runtime API

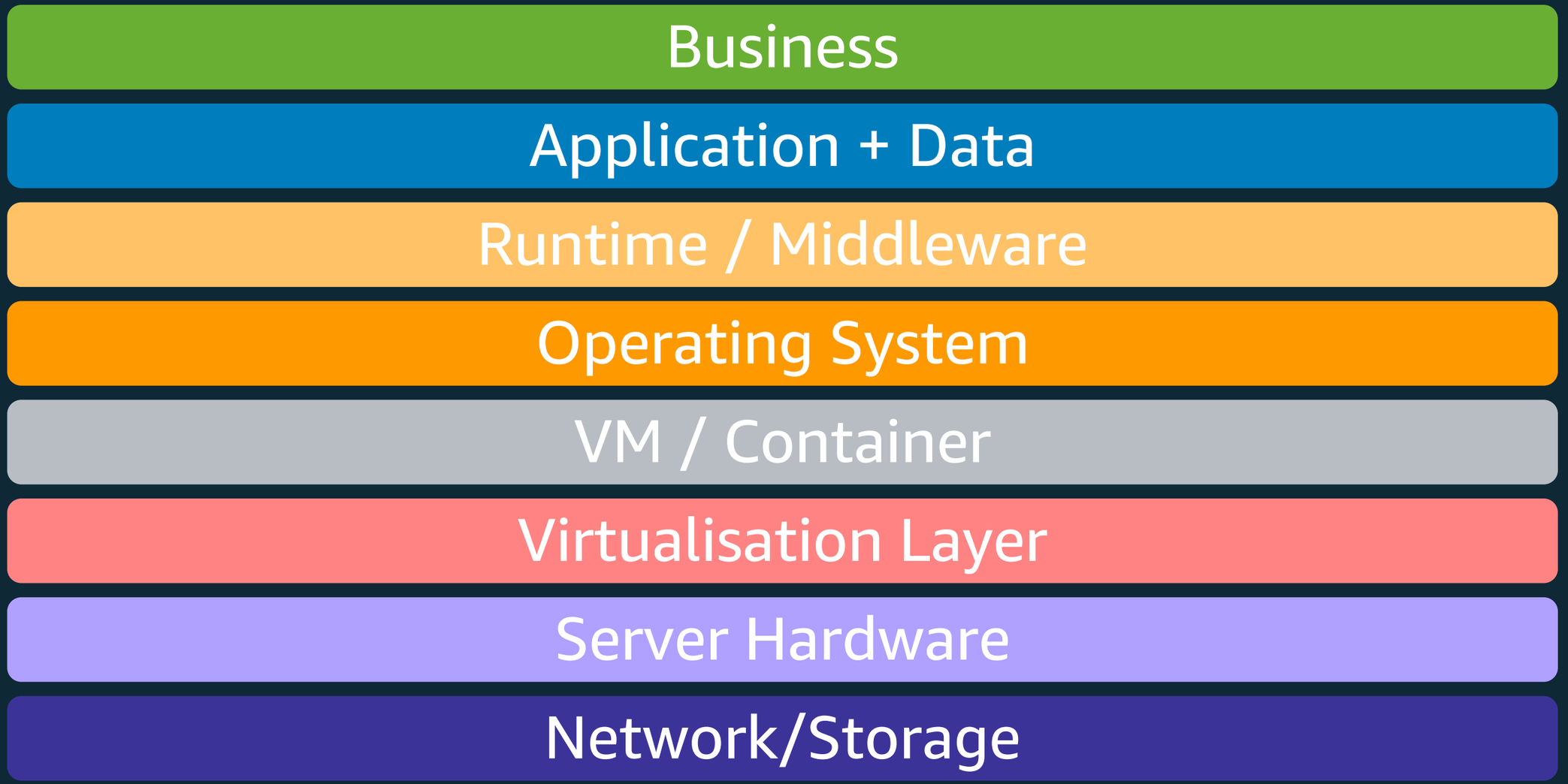
Services (anything)

aws

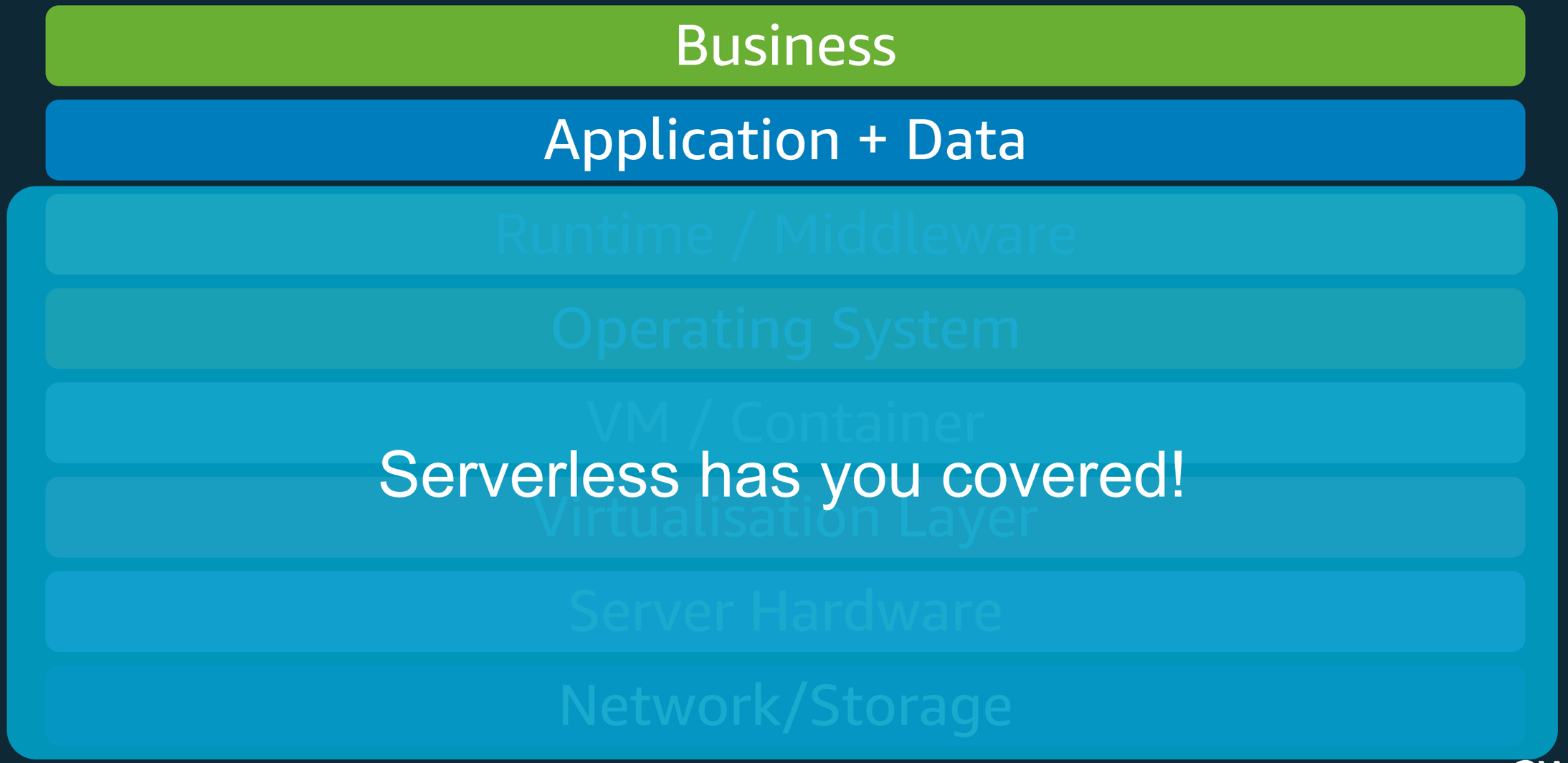
A serverless web application architecture



Traditional application stack layers



Traditional application stack layers



What is observability?

More than monitoring failures

Is it behaving as expected?

What is the usage?

What's the business impact?

More than monitoring failures

Observability is more about your organizational culture and practices than it is your tooling

Three pillars of observability tooling

Metrics	Logs	Traces
Numeric data measured at various time intervals (time series data); SLIs (request rate, error rate, duration, CPU%, etc.)	Timestamped records of discrete events that happened within an application or system, such as a failure, an error, or a state transformation	A trace represents a single user's journey across multiple applications and systems (usually microservices)

Definitions from: Distributed Systems Observability
https://www.oreilly.com/library/view/distributed-systems-observability/9781492033431/

aws

AWS services for observability

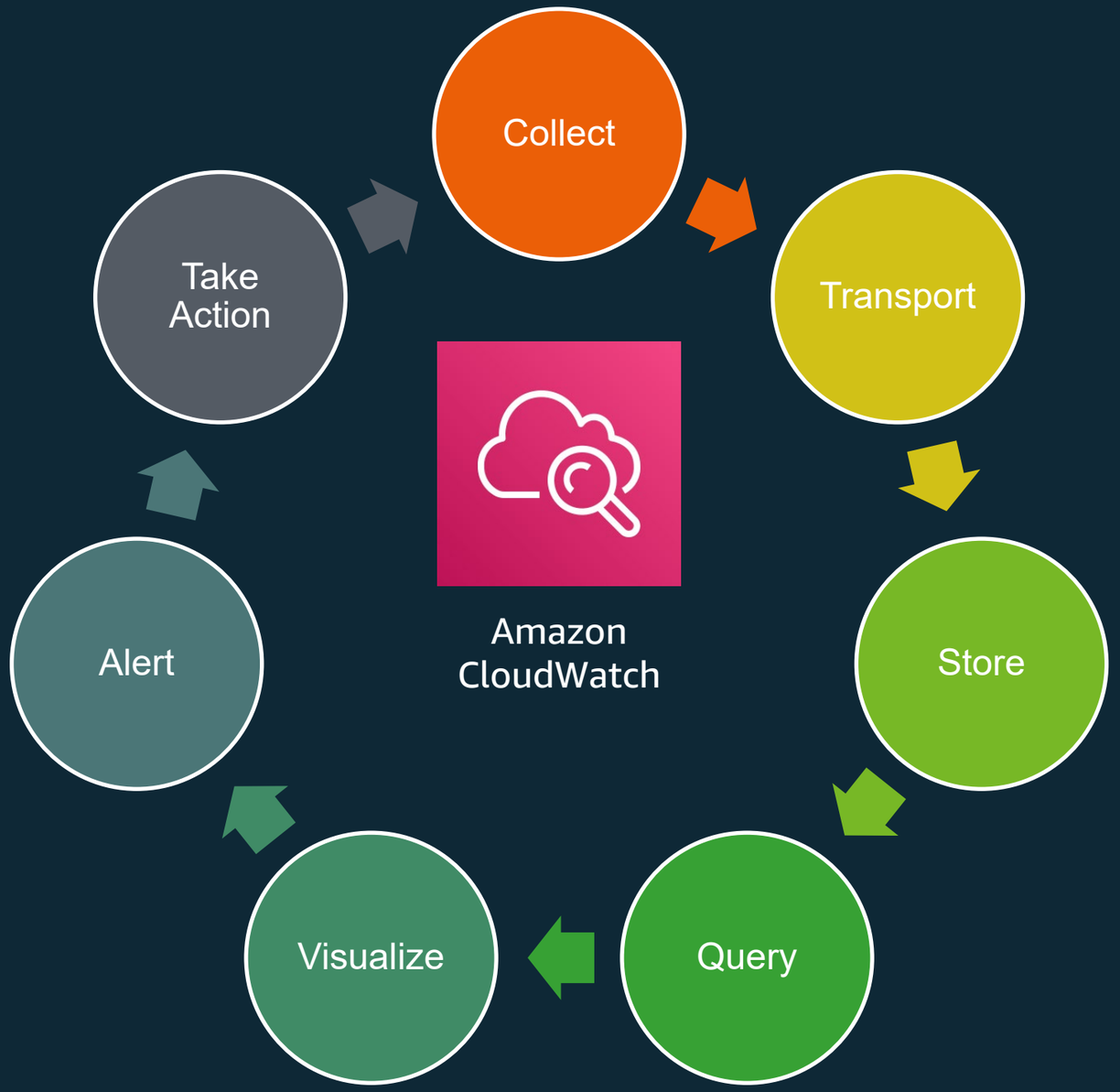
Amazon CloudWatch

AWS X-Ray

Dashboards
Logs
Metrics
Alarms
Events

Traces
Analytics
Service Map

Amazon CloudWatch can…



Amazon CloudWatch

1 Quadrillion +

(1,000,000,000,000,000 +)

Metric observations each month

3.9 Trillion

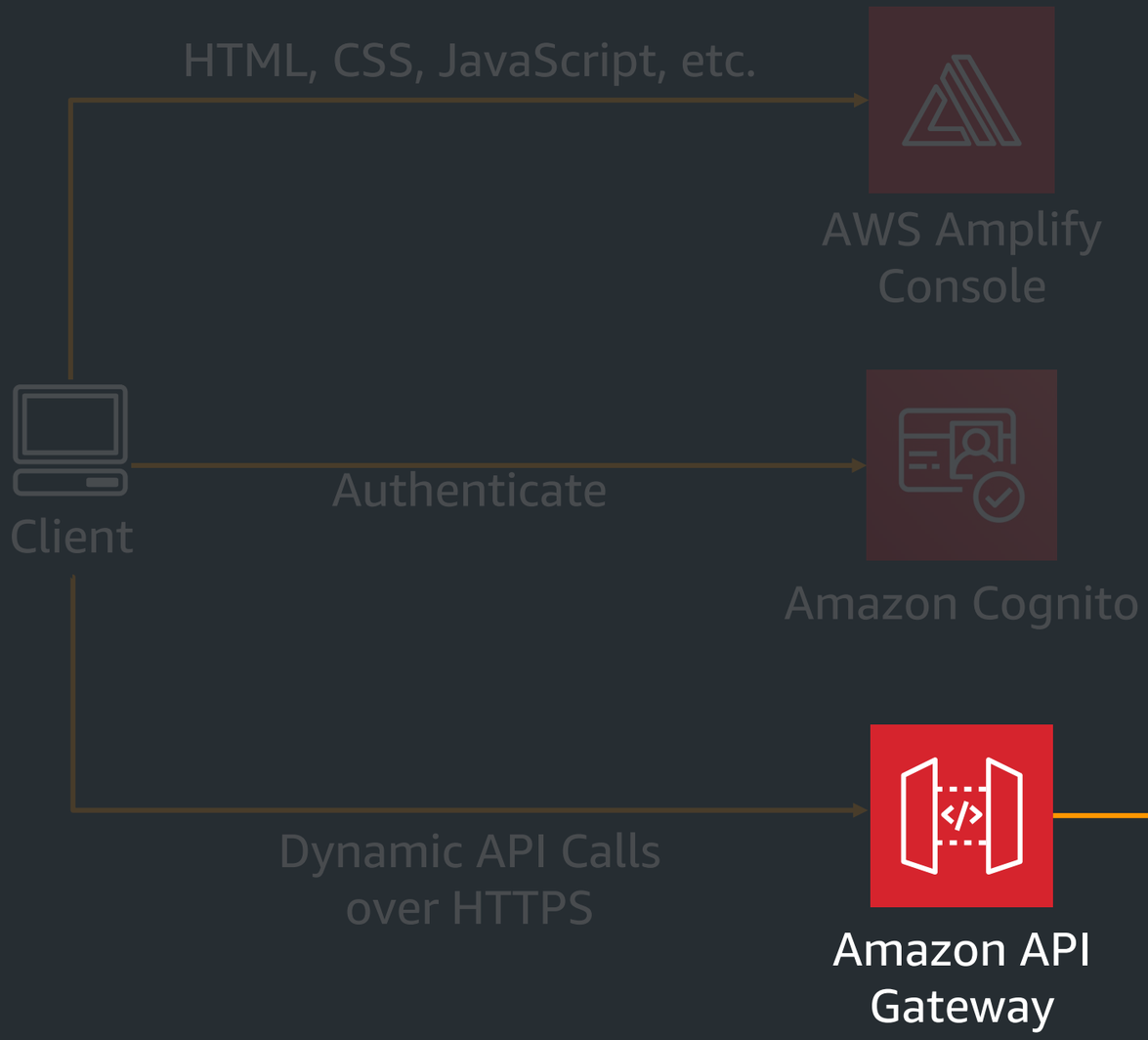
Events each month

Monitors entire infrastructure of AWS and Amazon.com

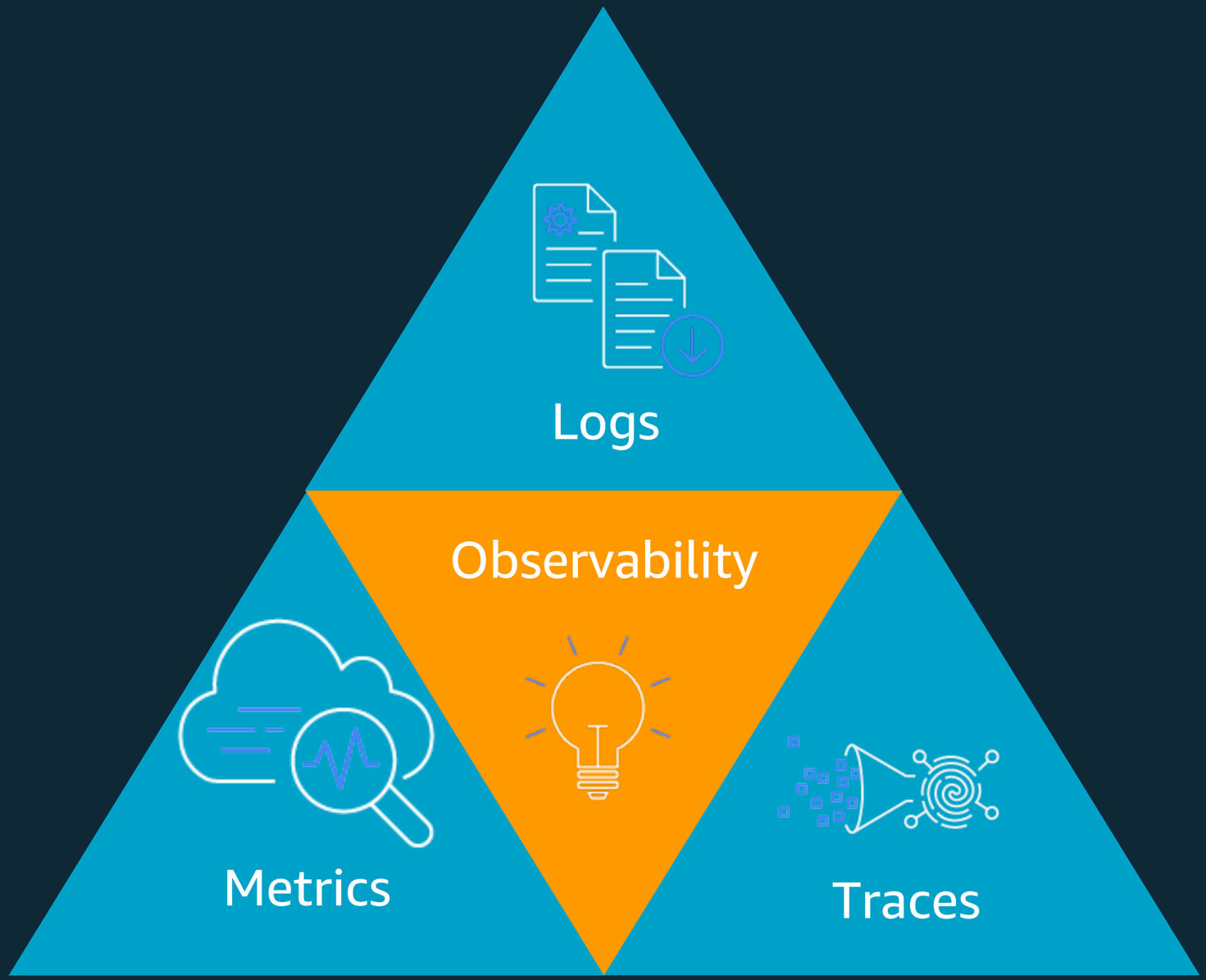
100 PB

Logs ingested each month

aws



Let’s drill into 3 of the services here:



CloudWatch Metrics

Built-in metrics

Amazon API Gateway

4XXError, 5XXError, CacheHitCount, CacheMissCount, Count, IntegrationLatency, Latency

Amazon CloudWatch

AWS Lambda

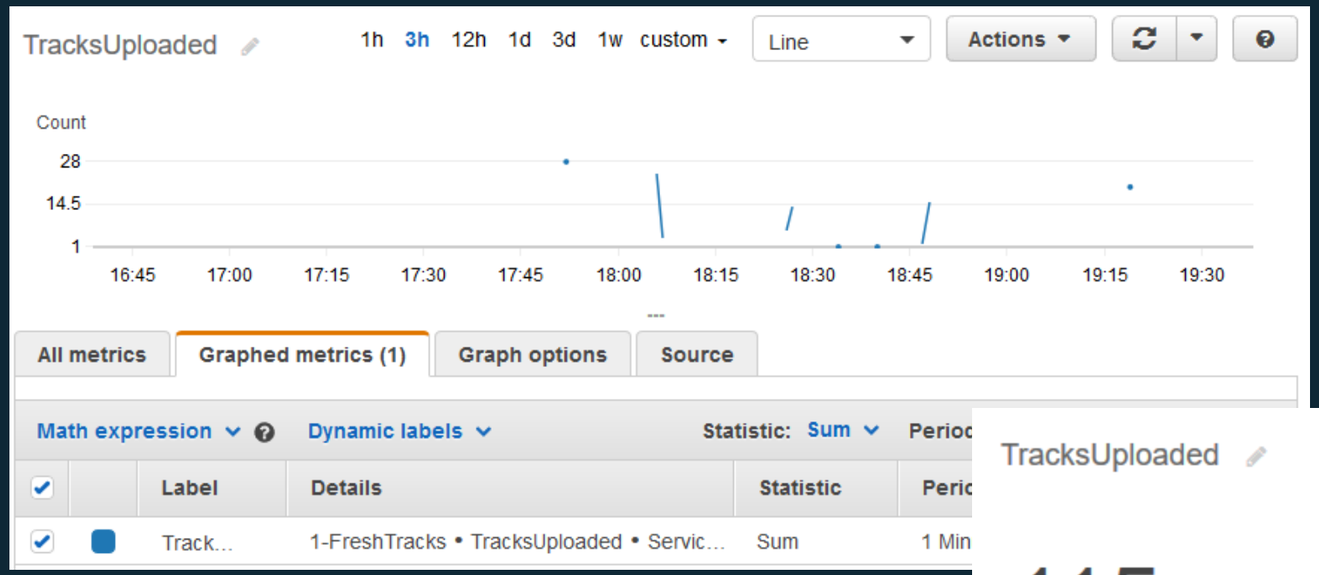
Invocations, Errors, DeadLetterErrors, Duration, Throttles, IteratorAge, ConcurrentExecutions, UnreservedConcurrentExecutions, ProvisionedConcurrentExecutions, ProvisionedConcurrencyInvocations, ProvisionedConcurrencySpilloverInvocations, ProvisionedConcurrencyUtilization

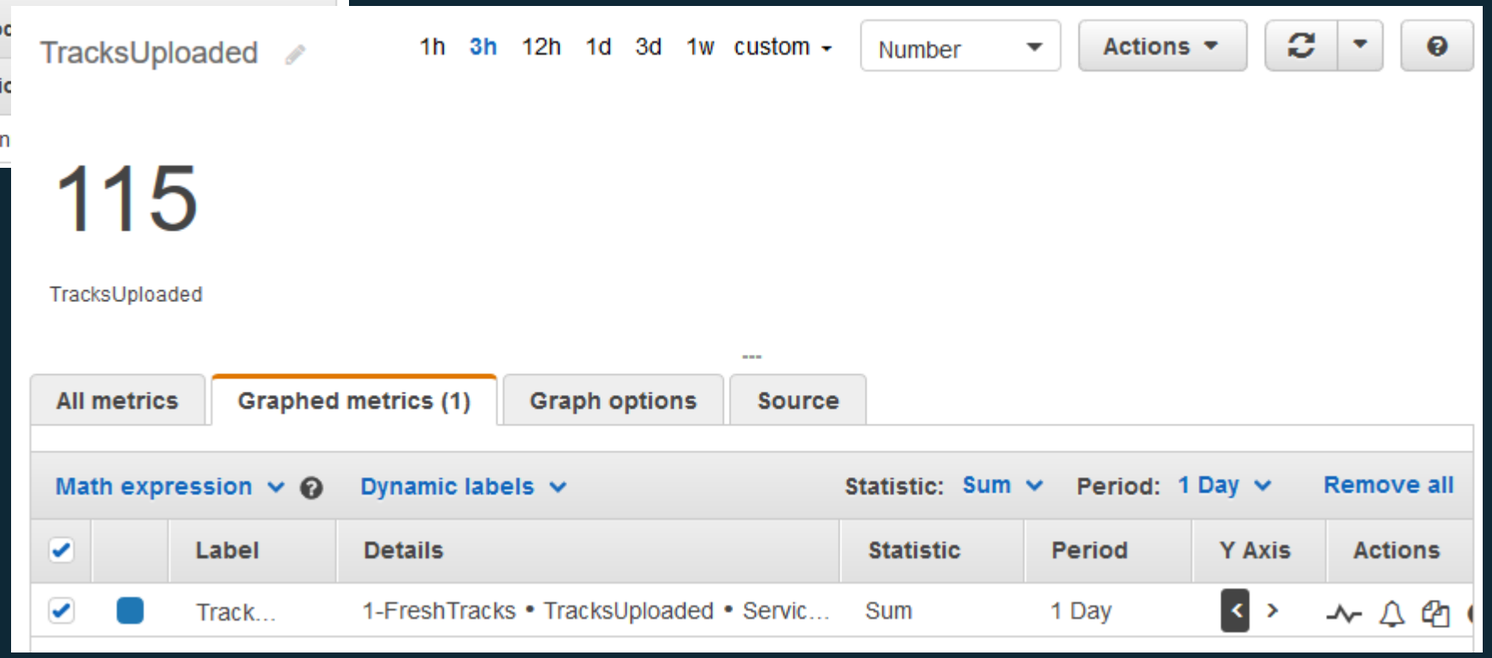
Built-in metrics

Amazon DynamoDB

AccountMaxReads, AccountMaxTableLevelReads, AccountMaxTableLevelWrites, AccountMaxWrites, AccountProvisionedReadCapacityUtilization, AccountProvisionedWriteCapacityUtilization, ConditionalCheckFailedRequests, ConsumedReadCapacityUnits, ConsumedWriteCapacityUnits, MaxProvisionedTableReadCapacityUtilization, MaxProvisionedTableWriteCapacityUtilization, OnlineIndexConsumedWriteCapacity, OnlineIndexPercentageProgress, OnlineIndexThrottleEvents, PendingReplicationCount, ProvisionedReadCapacityUnits, ProvisionedWriteCapacityUnits, ReadThrottleEvents, ReplicationLatency, ReturnedBytes, ReturnedItemCount, ReturnedRecordsCount, SuccessfulRequestLatency, SystemErrors, TimeToLiveDeletedItemCount, ThrottledRequests, TransactionConflict, UserErrors, WriteThrottleEvents

Visualize with CloudWatch Metrics Graphs





That's often not enough

What about caught errors? Warnings?

Caught exceptions are not counted as Errors on AWS Lambda.

What about business-relevant metrics?

Revenue, page views, etc.

What if I want to use other dimensions?

User ID, category, item, tags, environment, etc.

Creating custom metrics

Useful for: application, business, and operations metrics

- Use built in capabilities of the AWS SDK to call the CloudWatch “putMetricData” API Call
- Charged by metric and by put call of data into a metric

Improved upon by Embedded Metrics Format (covered shortly)

```
const AWS = require("aws-sdk")
AWS.config.region = ( process.env.AWS_REGION )
const cloudwatch = new AWS.CloudWatch({apiVersion: '2010-08-01'});

exports.handler = async (event) => {
  let params = { MetricData: [], Namespace: 'InnovatorIsland'}
  // Incoming message
  const msg = JSON.parse(event.detail.msg)

  msg.map((stat) => {
    params.MetricData.push({
      'MetricName': 'wait-times',
      'Dimensions': [
        {'Name': 'Type','Value': 'ride'},
        {'Name': 'Ride','Value': stat.rideId},
      ],
      'Unit': 'Seconds','Value': (stat.wait * 60)}
    )
  })
  // Send to CloudWatch
  console.log(await cloudwatch.putMetricData(params).promise())
}
```


aws

CloudWatch Logging

Built-in logging

API Gateway Logging

- 2 Levels of logging, ERROR and INFO
- Optionally log method request/body content
- Set globally in stage, or override per method

Lambda Logging

- Logging directly from your code with your language's equivalent of `console.log()` - basic request information included
- **Can use** `PutLogEvents` AWS CloudWatch SDK call to put custom log information with more structure

Log Pivots

- Build metrics based on log filters
- Jump to logs that generated metrics

Export logs to AWS ElastiCache or S3

- Explore with Kibana or Amazon Athena/Amazon QuickSight

Amazon CloudWatch

aws

Structured logging

```
message = {
  "PriceInCart": 100,
  "QuantityInCart": 2,
  "ProductId": "a23390f3",
  "CategoryId": "bca4cec1",
  "UserId": "31ba3930",
  "CartId": "58dd189f",
  "Environment": "prod",
  "LogLevel": "INFO",
  "Timestamp": "2019-12-11 12:44:40.300473",
  "Message": "Added 2 items 'a23390f3' to cart
'58dd189f'"
}
```

```
# Python
print(json.dumps(message))

// Javascript
console.log(JSON.stringify(message));

// Go
enc := json.NewEncoder(os.Stdout)
enc.Encode(message)

// Java
class JSONLoggerInitializerFactory...
```


aws

CloudWatch Embedded metrics format

Automatically generate metrics from structured CloudWatch Logs

- One call gets you both
- Open-source client libraries available for Node.js and Python make it easy
- Can send structured format in normal `PutLogEvents` call with specific format

Installation

```
pip3 install aws-embedded-metrics
```

Usage

To get a metric logger, you can decorate your function with a `metric_scope`:

```
from aws_embedded_metrics import metric_scope

@metric_scope
def my_handler(metrics):
    metrics.put_dimensions({"Foo": "Bar"})
    metrics.put_metric("ProcessingLatency", 100, "Milliseconds")
    metrics.set_property("AccountId", "123456789012")
    metrics.set_property("RequestId", "422b1569-16f6-4a03")
    metrics.set_property("DeviceId", "61270781-c6ac-46f1")

    return {"message": "Hello!"}
```

Embedded Metric Format

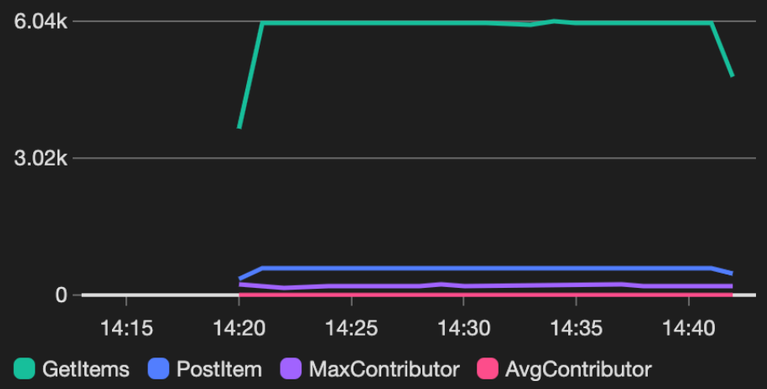
```
message = {
  "PriceInCart": 100,
  "QuantityInCart": 2,
  "ProductId": "a23390f3",
  "CategoryId": "bca4cec1",
  "UserId": "31ba3930",
  "CartId": "58dd189f",
  "Environment": "prod",
  "LogLevel": "INFO",
  "Timestamp": "2019-12-11 12:44:40.300473",
  "Message": "Added 2 items 'a23390f3' to cart
'58dd189f'"
}
```

```
  [...]
  "_aws": {
    "Timestamp": 1576064416496,
    "CloudWatchMetrics": [{
      "Namespace": "ecommerce-cart",
      "Dimensions": [
        ["Environment", "CategoryId"]
      ],
      "Metrics": [
        {"Name": "PriceInCart", "Unit": "None"},
        {"Name": "QuantityInCart", "Unit": "None"}
      ]
    }]
  }
}
```


aws

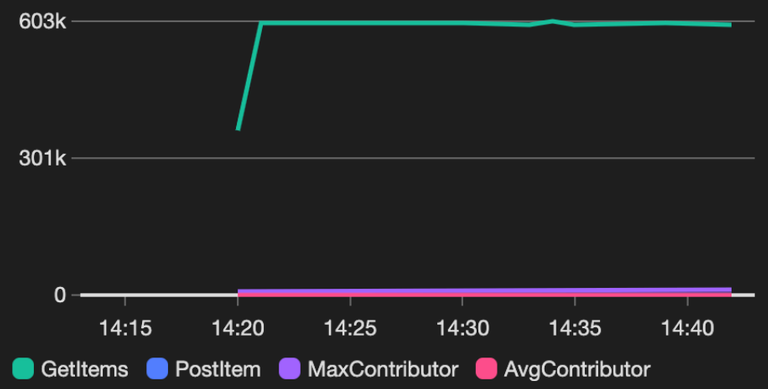
Requests

Various units



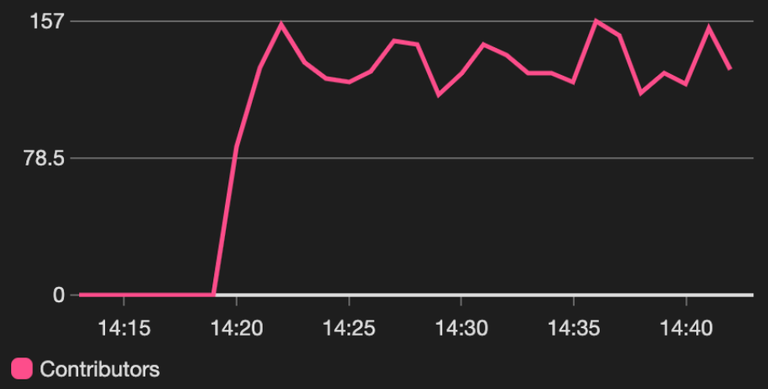
Items

No unit



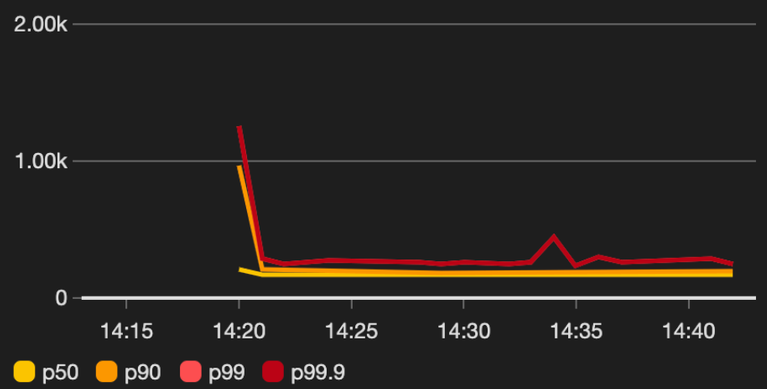
Number of contributors

No unit



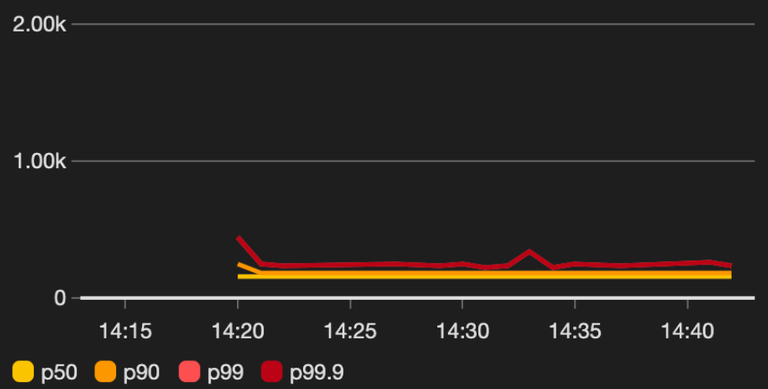
API Gateway Latency

Milliseconds



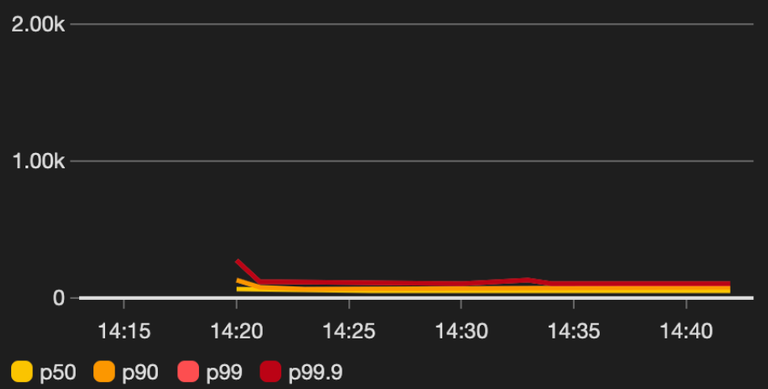
GetItems Duration

Milliseconds



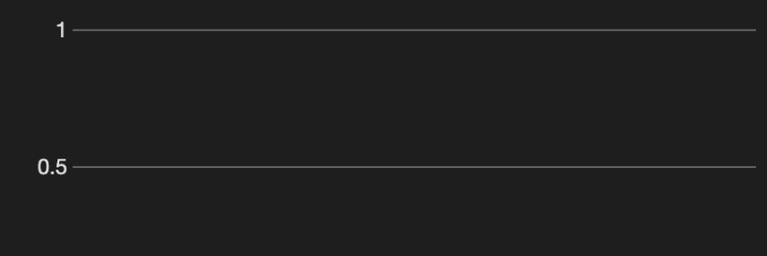
PostItem Duration

Milliseconds



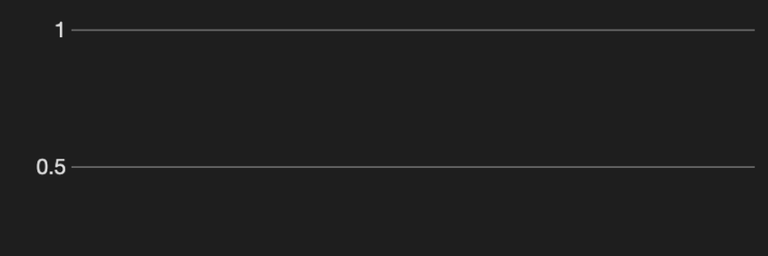
Backend Errors

Count



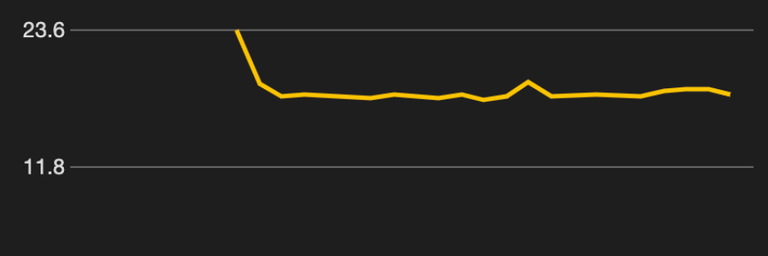
Throttles

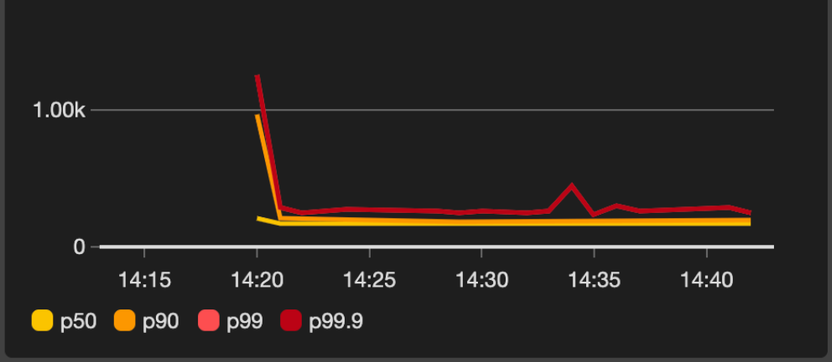
Count

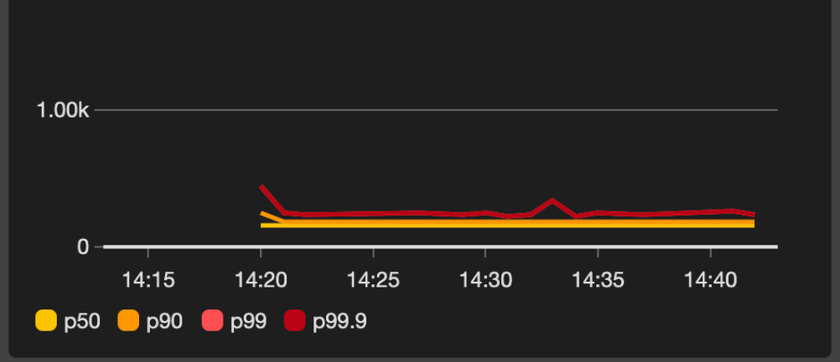


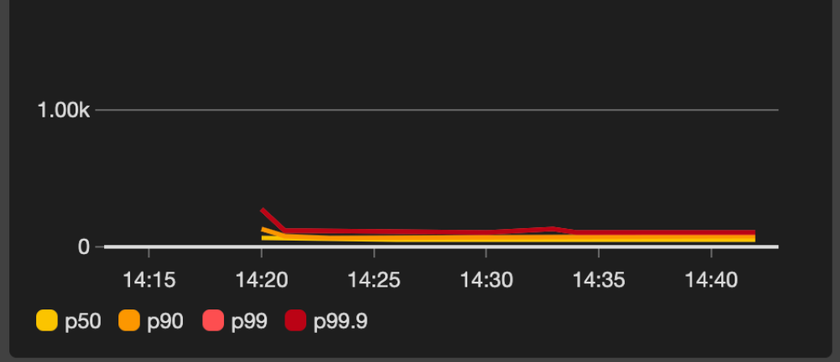
Concurrent Executions

Count

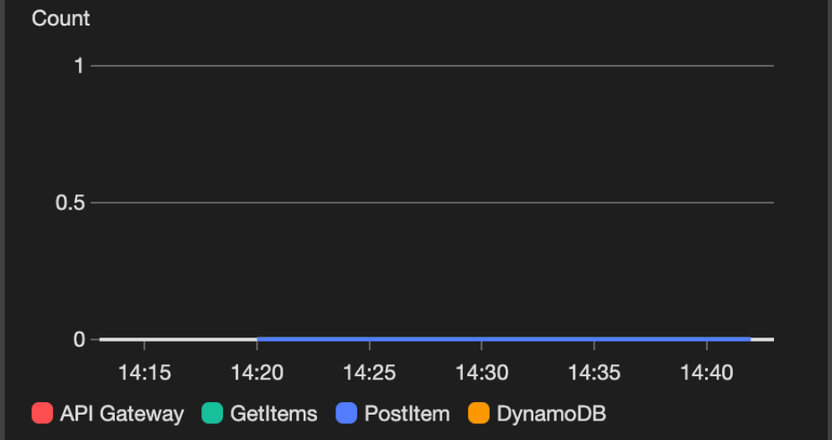




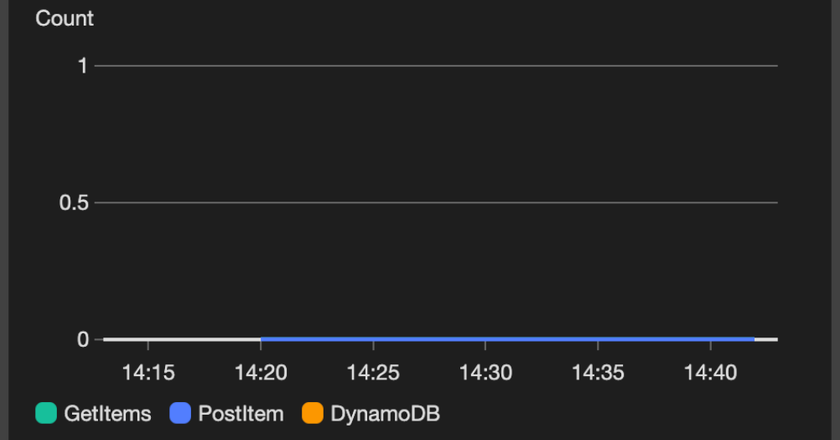




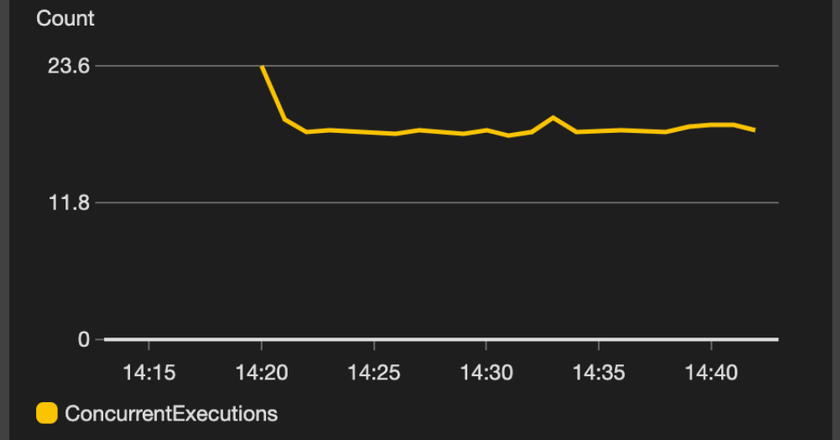
Backend Errors



Throttles



Concurrent Executions



Log group: multiple (2)

#	@timestamp	loglevel	message	storeId
1	2019-12-11T14:42:31.868Z	ERR	Hash value mismatch	109aa45d-b7f2-5f8a-9650-79bee04a29dc
2	2019-12-11T14:42:30.470Z	ERR	Hash value mismatch	49eb952a-b1e7-57ea-a234-5e1f99dbef38
3	2019-12-11T14:42:30.470Z	ERR	Hash value mismatch	49eb952a-b1e7-57ea-a234-5e1f99dbef38
4	2019-12-11T14:42:30.328Z	ERR	Hash value mismatch	49eb952a-b1e7-57ea-a234-5e1f99dbef38
5	2019-12-11T14:42:30.130Z	ERR	Hash value mismatch	49eb952a-b1e7-57ea-a234-5e1f99dbef38
6	2019-12-11T14:42:29.989Z	ERR	Hash value mismatch	49eb952a-b1e7-57ea-a234-5e1f99dbef38
7	2019-12-11T14:42:29.790Z	ERR	Hash value mismatch	49eb952a-b1e7-57ea-a234-5e1f99dbef38
8	2019-12-11T14:42:29.450Z	ERR	Hash value mismatch	d9494534-922f-5aa6-89a4-d00321fd3322
9	2019-12-11T14:42:29.103Z	ERR	Hash value mismatch	d9494534-922f-5aa6-89a4-d00321fd3322
10	2019-12-11T14:42:28.570Z	ERR	Hash value mismatch	456b8b15-dd3d-5a41-bfc1-bc7388437c0b

Amazon CloudWatch Logs Insights

Drive actionable intelligence from your logs to address operational issues without needing to provision servers or manage software.

Get the last 100 error messages

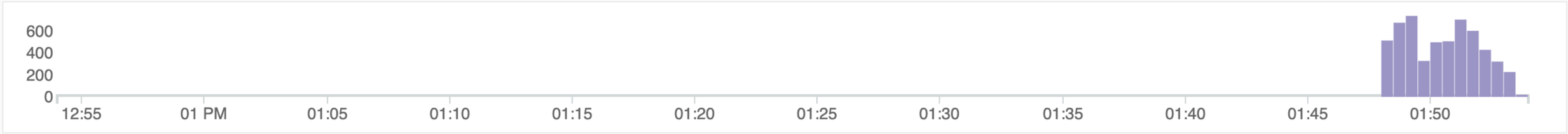

```
{
  "ProductId": "a23390f3",
  "CategoryId": "bca4cec1",
  "UserId": "31ba3930",
  "CartId": "58dd189f",
  "Environment": "prod",
  "LogLevel": "ERR",
  "Timestamp": "2019-12-11 12:44:40.300473",
  "Message": "Failed to add 'a23390f3' to cart
'58dd189f'"
}
```


```
fields Timestamp, LogLevel, Message
| filter LogLevel == "ERR"
| sort @timestamp desc
| limit 100
```


aws

Get the last 100 error messages

Logs | Visualization



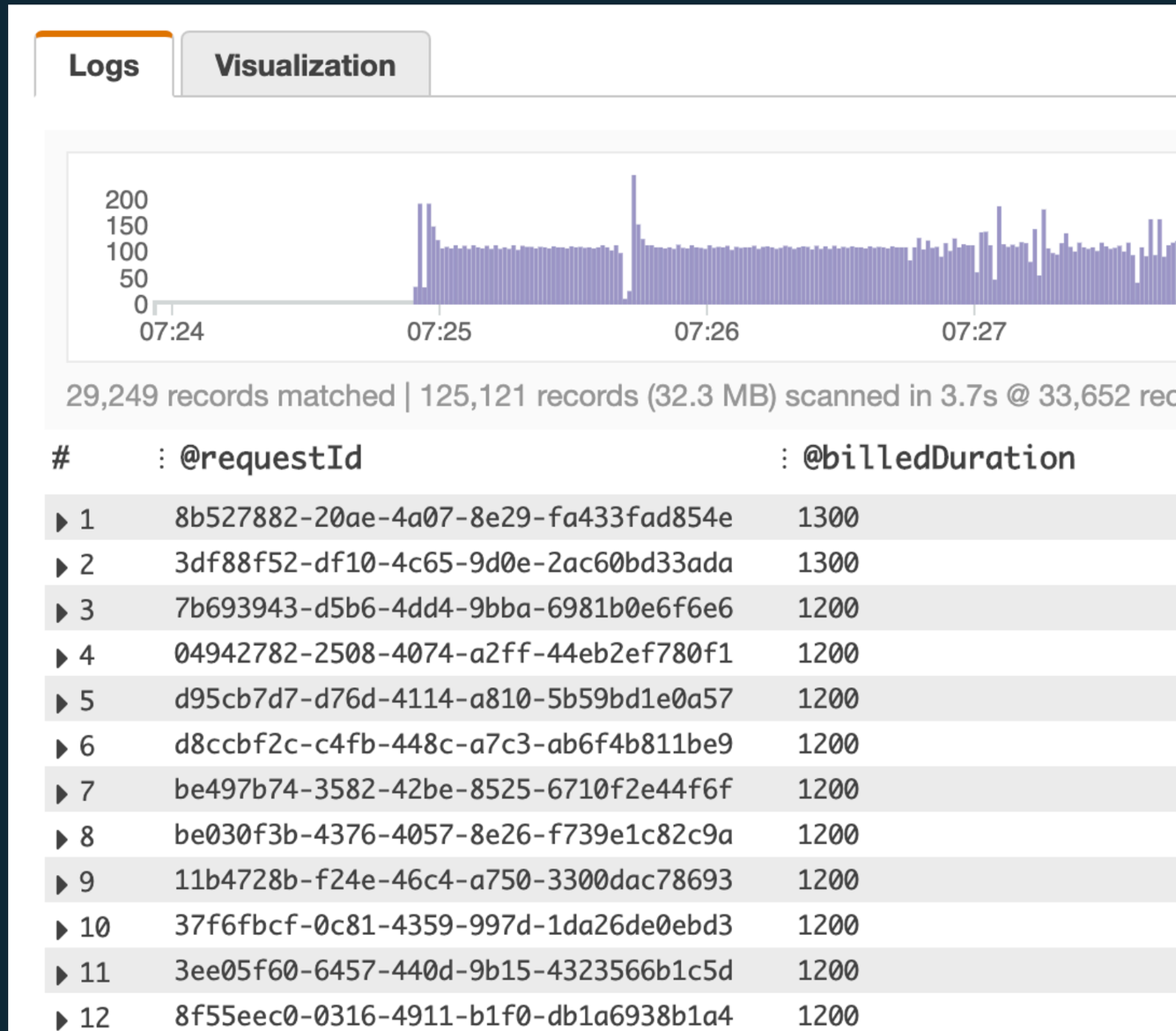
5,633 records matched | 131,053 records (33.8 MB) scanned in 3.3s @ 40,016 records/s (10.3 MB/s)

#	@timestamp	loglevel	message
▸ 1	2019-12-11T13:53:43.870+01:00	ERR	Hash value mismatch
▸ 2	2019-12-11T13:53:43.690+01:00	ERR	Hash value mismatch
▾ 3	2019-12-11T13:53:41.950+01:00	ERR	Hash value mismatch

@ingestionTime	1576068829802
@log	017024781235:/aws/lambda/demo-tko-develop-GetItemsFunction-73FAT6Z12SPP
@logStream	2019/12/11/[$LATEST]e309f7494d944c8ba4d5511225346ee1
@message	{"operation": "get_items", "storeId": "49eb952a-b1e7-57ea-a234-5e1f99dbef38", "itemId": "09063559-cb29-482d-8a08-e6bfe162201f", "message": "
@timestamp	1576068821950
@xrayTraceId	1-5df0e6d5-f83793e0eb1202e01eab4620
environment	develop
functionName	demo-tko-develop-GetItemsFunction-73FAT6Z12SPP
functionVersion	$LATEST
itemId	09063559-cb29-482d-8a08-e6bfe162201f

Top 100 most expensive executions

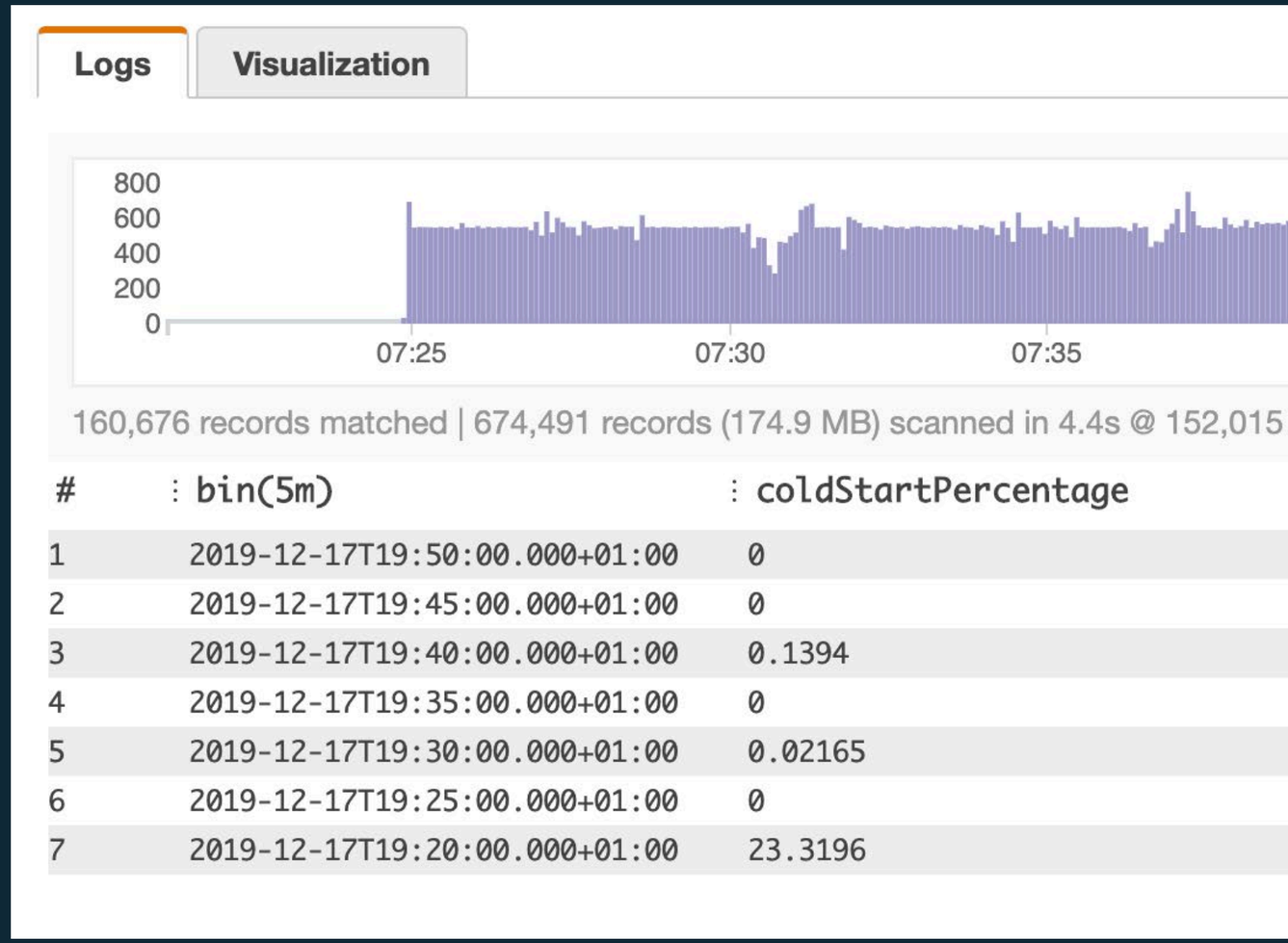
```
filter @type = "REPORT"
| fields @requestId, @billedDuration
| sort by @billedDuration desc
| limit 100
```



#	@requestId	@billedDuration
1	8b527882-20ae-4a07-8e29-fa433fad854e	1300
2	3df88f52-df10-4c65-9d0e-2ac60bd33ada	1300
3	7b693943-d5b6-4dd4-9bba-6981b0e6f6e6	1200
4	04942782-2508-4074-a2ff-44eb2ef780f1	1200
5	d95cb7d7-d76d-4114-a810-5b59bd1e0a57	1200
6	d8ccbf2c-c4fb-448c-a7c3-ab6f4b811be9	1200
7	be497b74-3582-42be-8525-6710f2e44f6f	1200
8	be030f3b-4376-4057-8e26-f739e1c82c9a	1200
9	11b4728b-f24e-46c4-a750-3300dac78693	1200
10	37f6fbcf-0c81-4359-997d-1da26de0ebd3	1200
11	3ee05f60-6457-440d-9b15-4323566b1c5d	1200
12	8f55eec0-0316-4911-b1f0-db1a6938b1a4	1200

Cold start percentage over time

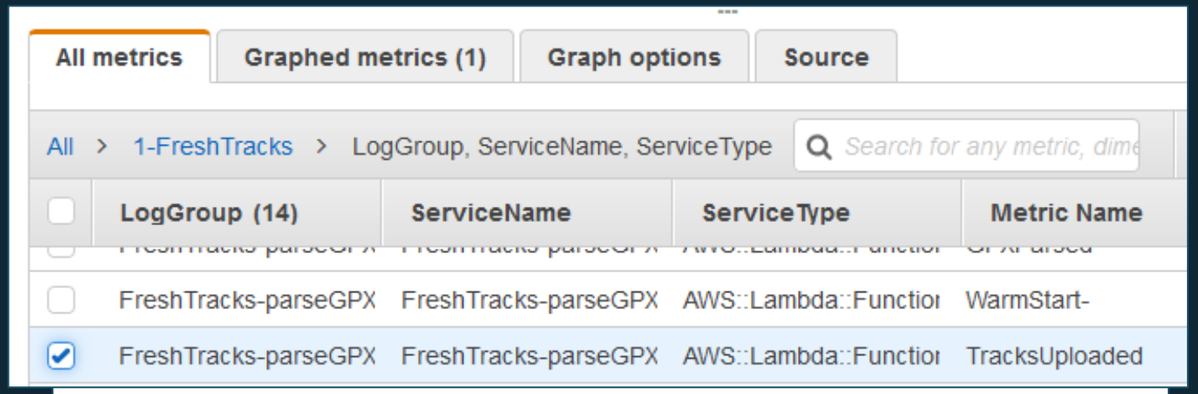
```
filter @type = "REPORT"
| stats
  sum(strcontains(
    @message,
    "Init Duration"))
  / count(*)
  * 100
  as coldStartPercentage,
  avg(@duration)
  by bin(5m)
```

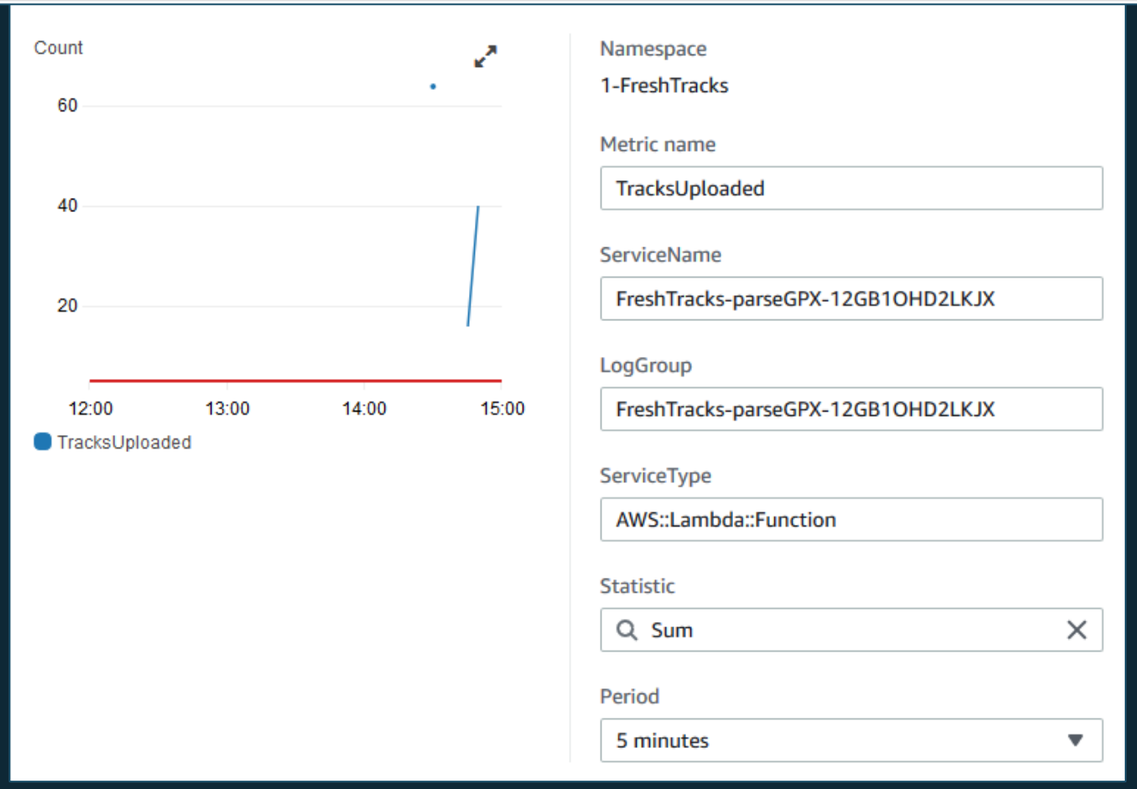


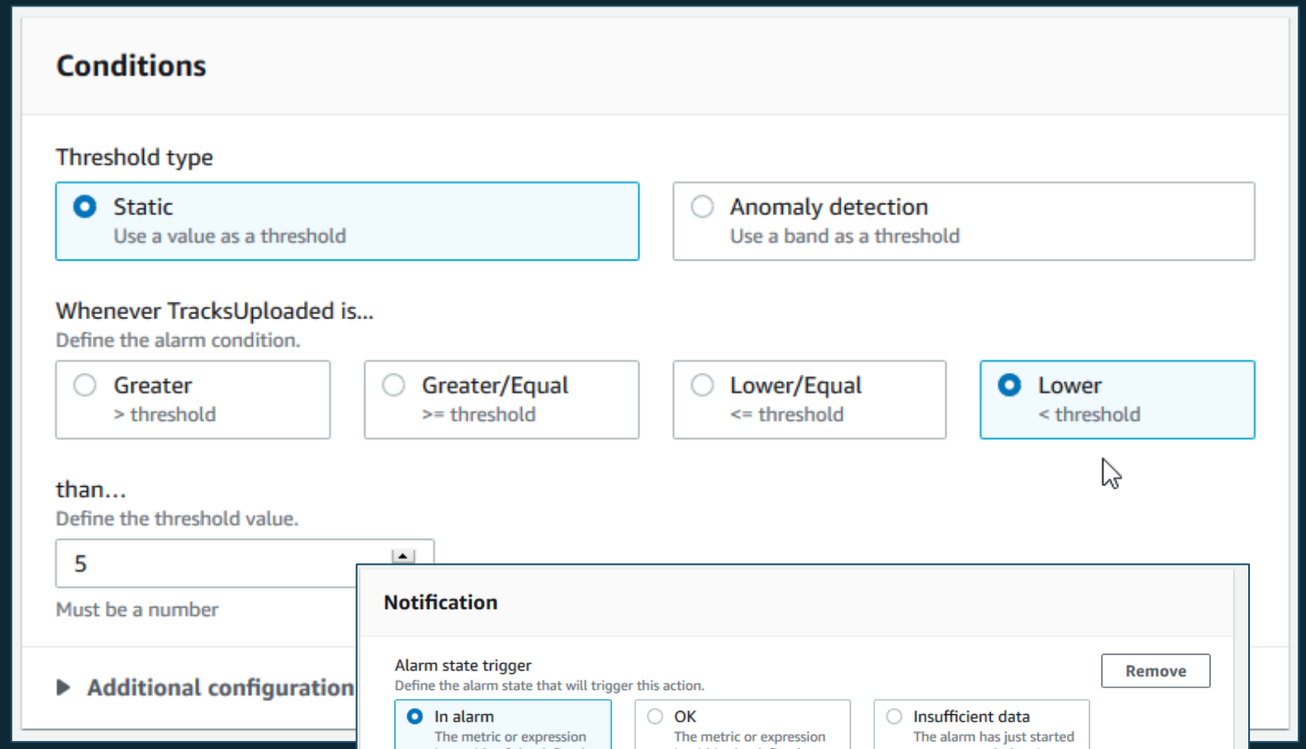
#	bin(5m)	coldStartPercentage
1	2019-12-17T19:50:00.000+01:00	0
2	2019-12-17T19:45:00.000+01:00	0
3	2019-12-17T19:40:00.000+01:00	0.1394
4	2019-12-17T19:35:00.000+01:00	0
5	2019-12-17T19:30:00.000+01:00	0.02165
6	2019-12-17T19:25:00.000+01:00	0
7	2019-12-17T19:20:00.000+01:00	23.3196

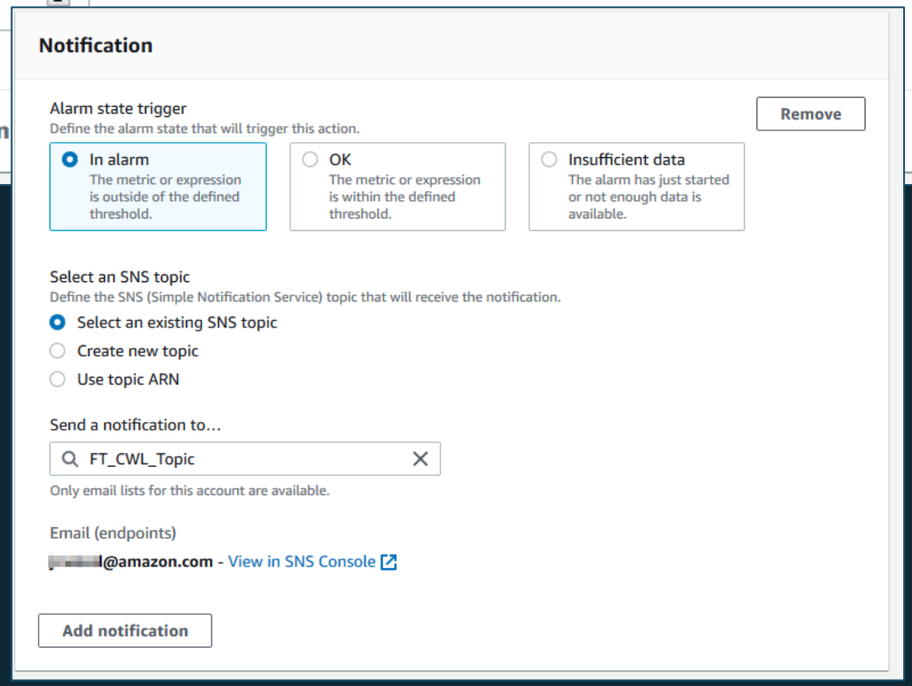
Creating alerts

Creating CloudWatch Alarms





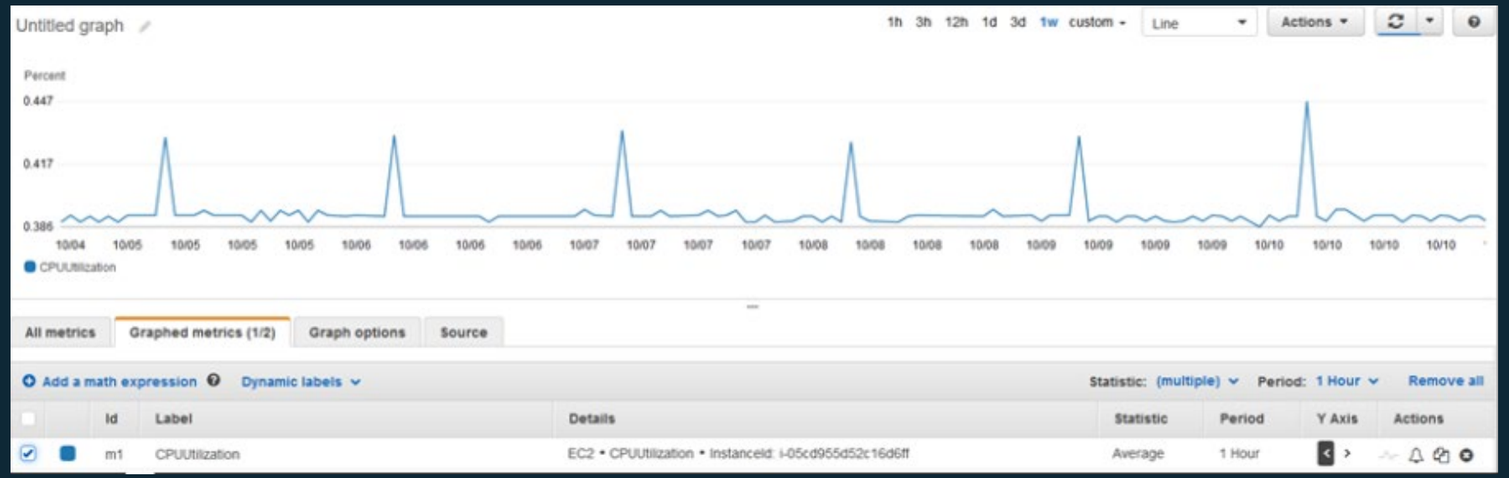


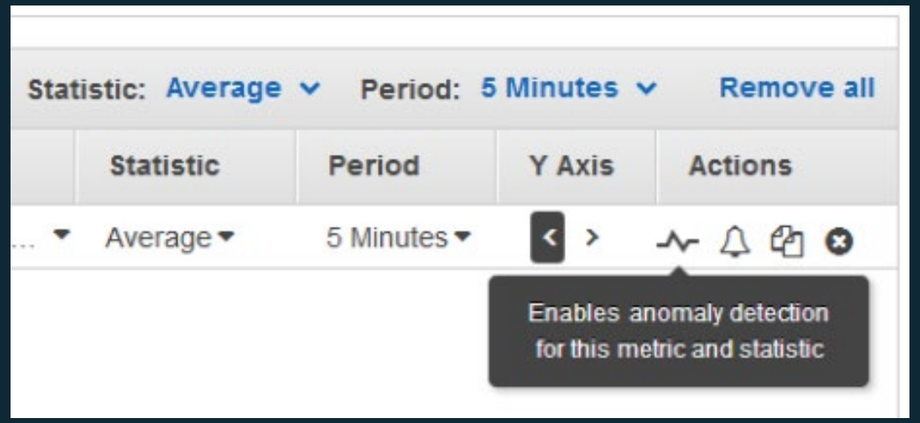


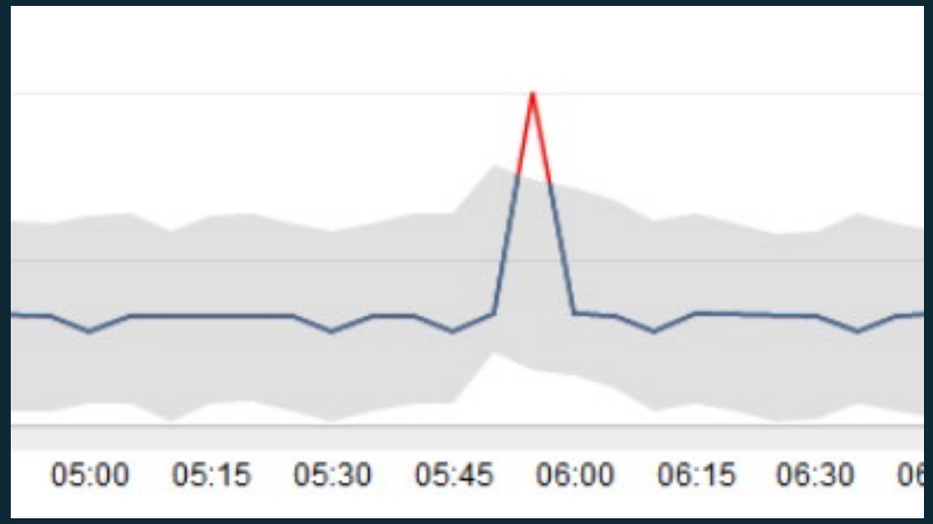
Configuring alerts via CloudFormation

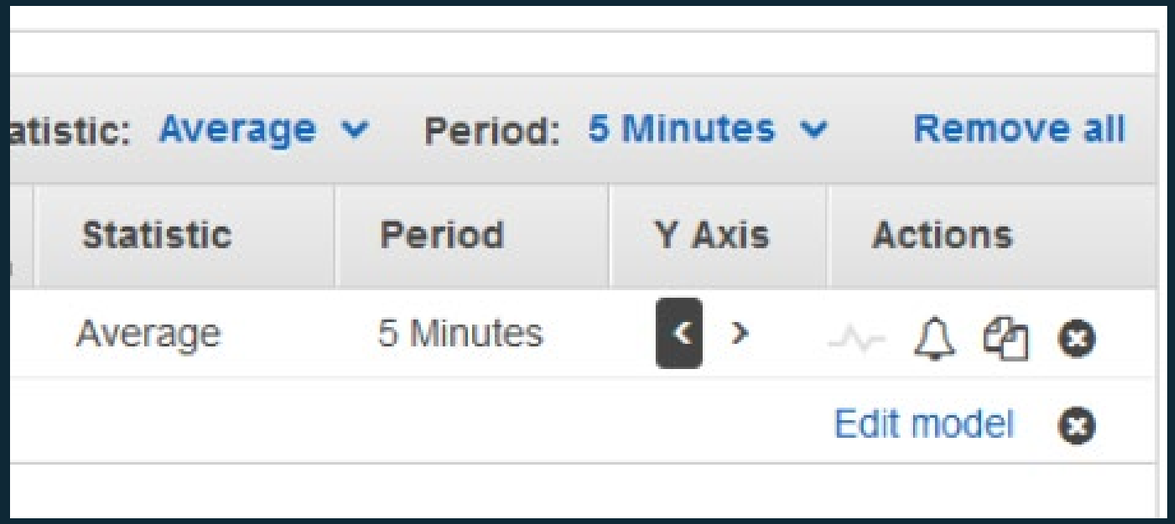
```yaml
P99ApiGatewayLatencyAlarm:
  Type: AWS::CloudWatch::Alarm
  Properties:
    AlarmDescription: P99 Latency for API Gateway
    ComparisonOperator: GreaterThanOrEqualToThreshold
    Dimensions:
      - Name: ApiName
        Value: !Ref AWS::StackName
    EvaluationPeriods: 1
    ExtendedStatistic: "p99"
    MetricName: "Latency"
    Namespace: "AWS/ApiGateway"
    Period: 60
    Threshold: 1000
    TreatMissingData: notBreaching
    Unit: Milliseconds
```

Using CloudWatch Anomaly Detection Alarms









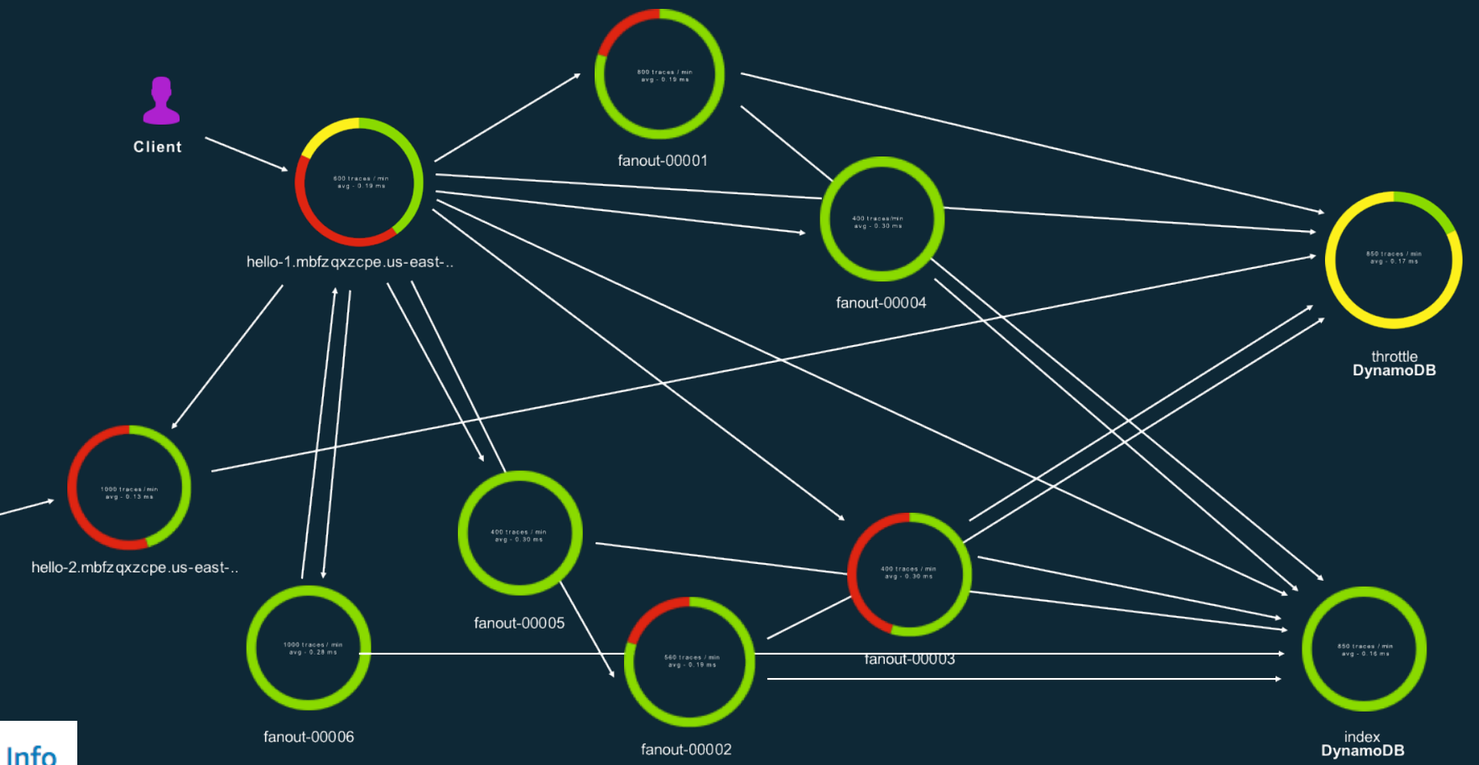
Tracing

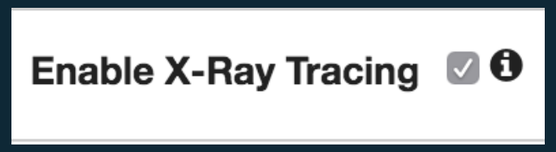
AWS X-Ray

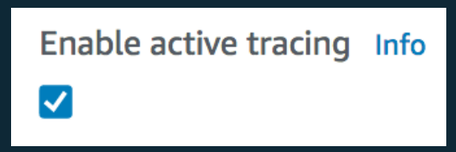
Profile and troubleshoot serverless applications:

- Lambda instruments incoming requests for all supported languages and can capture calls made in code
- API Gateway inserts a tracing header into HTTP calls as well as reports data back to X-Ray itself

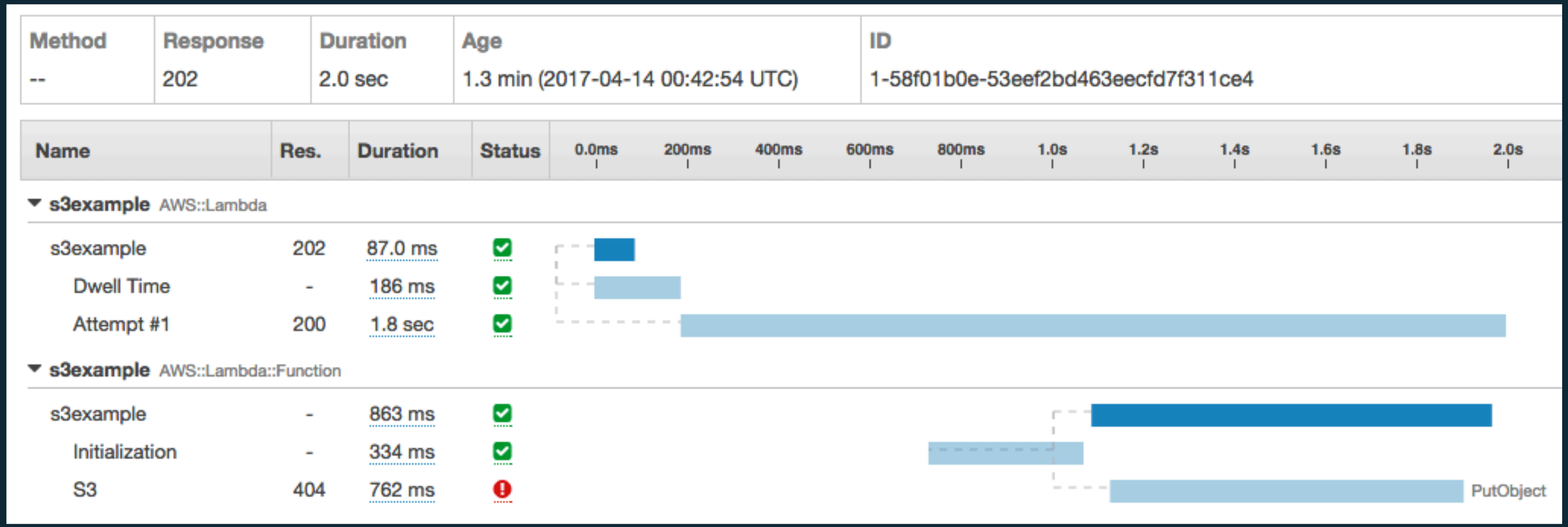
```
var AWSXRay = require('aws-xray-sdk-core');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
S3Client = AWS.S3();
```



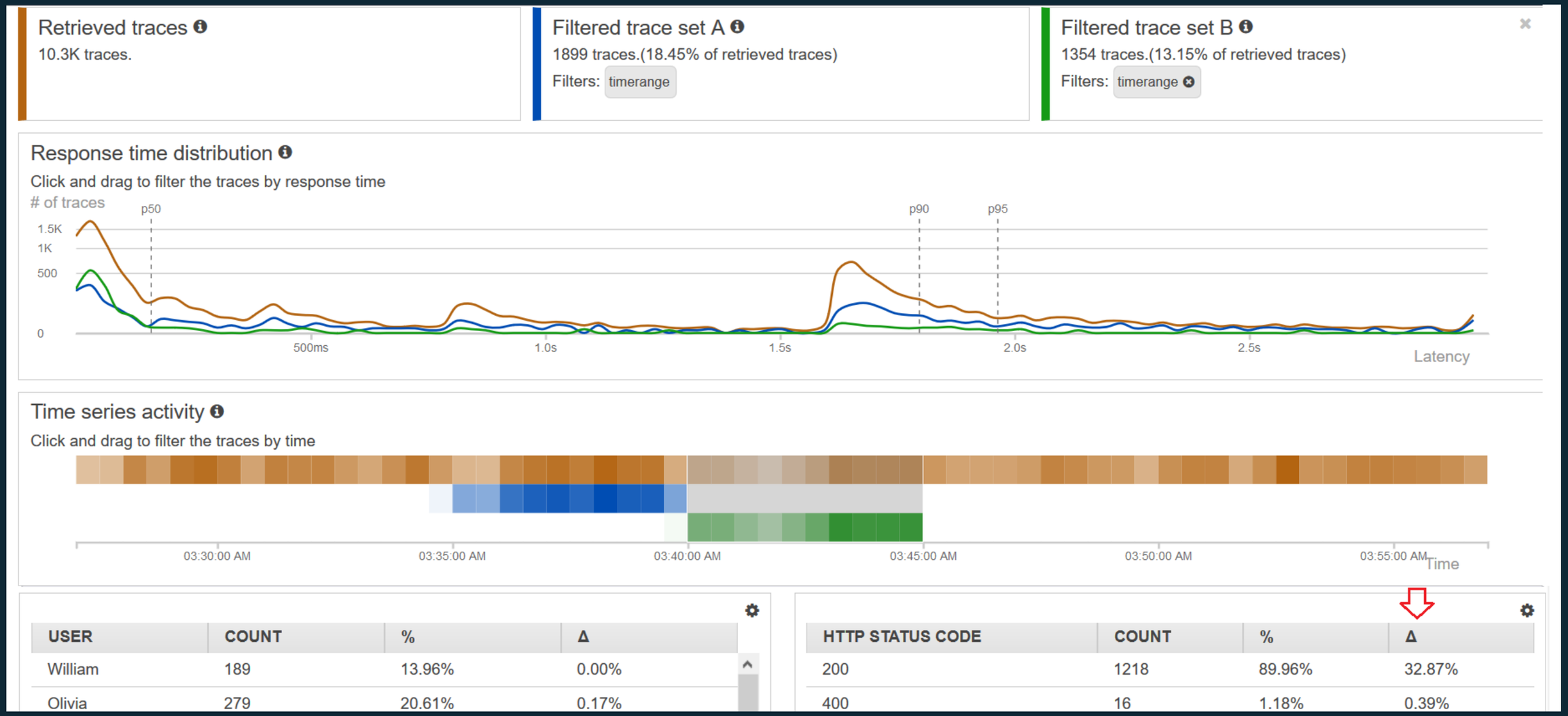




X-Ray Trace Example



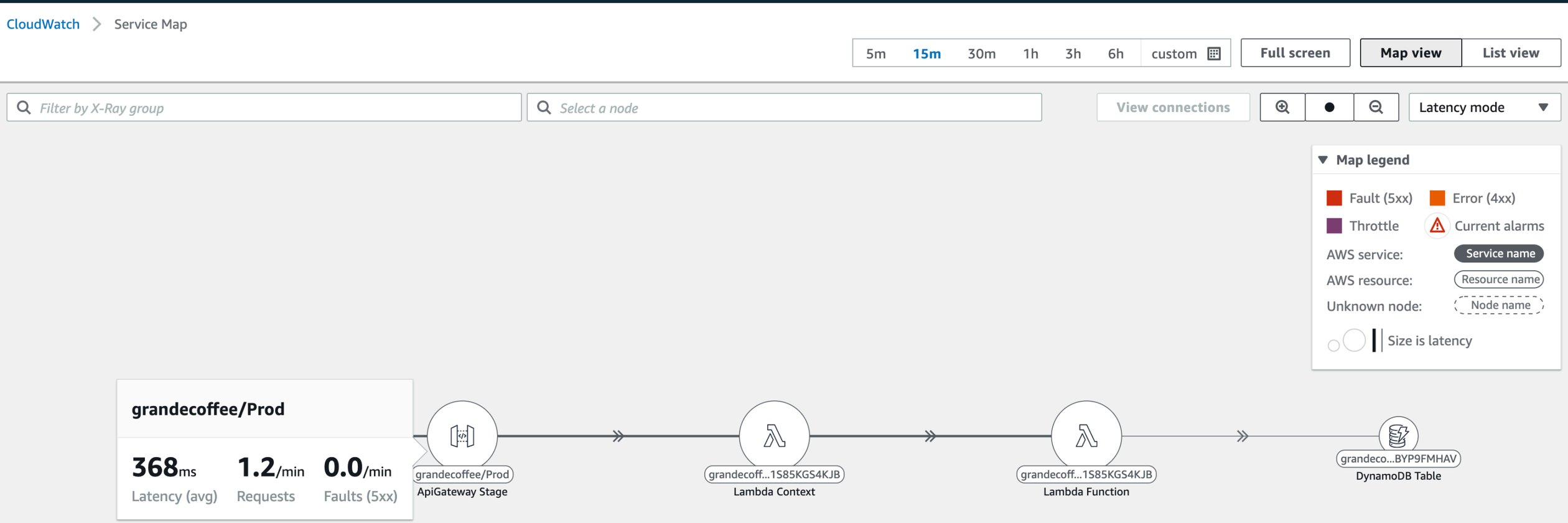
AWS X-Ray Analytics Example



CloudWatch ServiceLens

Ties together CloudWatch metrics and logs as well as traces from AWS X-Ray to give you a complete view of your applications and their dependencies.

CloudWatch ServiceLens

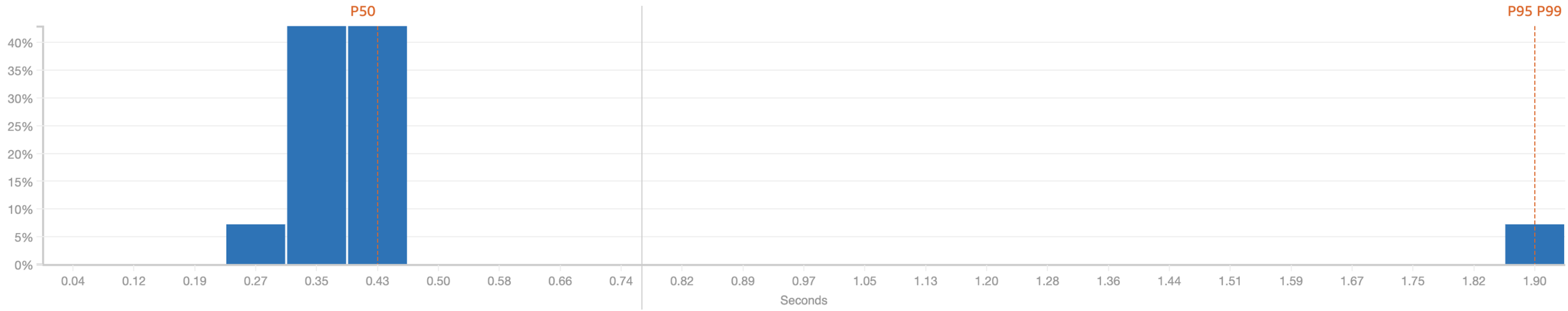


Visualize application performance

Response time distribution of traces

Select range to filter by response time

Add selected response times to filter



Traces (14)

View details

Find traces

1

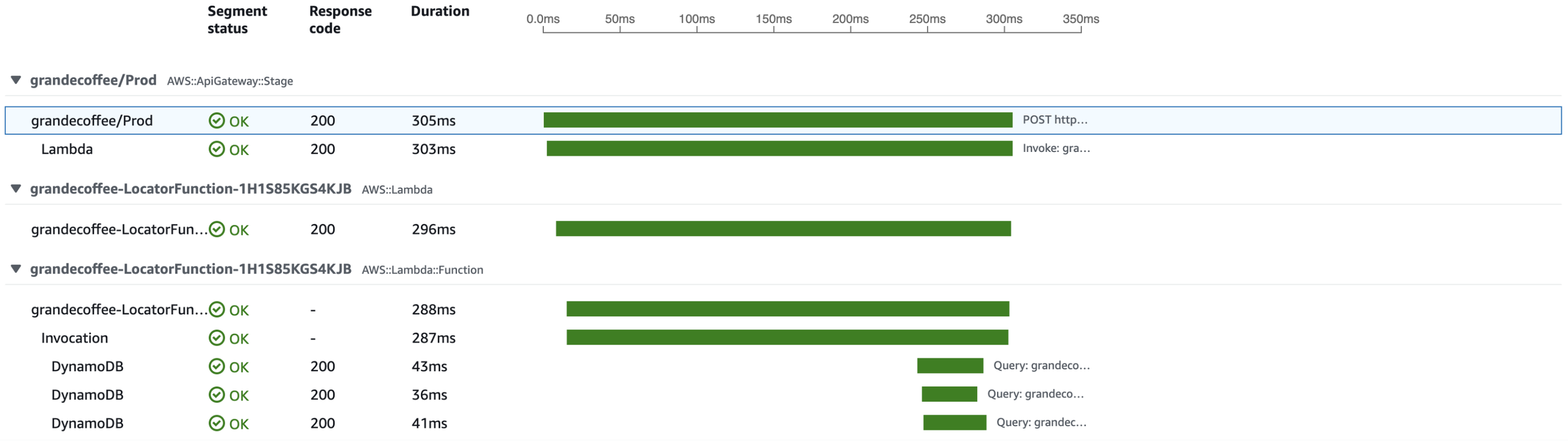
ID	Trace status	Timestamp	Response code	Response Time	HTTP Method	URL Address
1-5eb9abac-a3f8c157ffb1e049512e9828	OK	2.9min (2020-05-11 15:46:52)	200	0.305s	POST	https://dhx9sy1a4b.execute-api.us-east-2.amazonaws.com/Prod/
1-5eb9abde-34c0e3efde8dcd5e06171548	OK	2.1min (2020-05-11 15:47:42)	200	0.345s	POST	https://dhx9sy1a4b.execute-api.us-east-2.amazonaws.com/Prod/
1-5eb9aba2-1a2477b2318b7476d5d5d5b8	OK	3.1min (2020-05-11 15:46:42)	200	0.341s	POST	https://dhx9sy1a4b.execute-api.us-east-2.amazonaws.com/Prod/

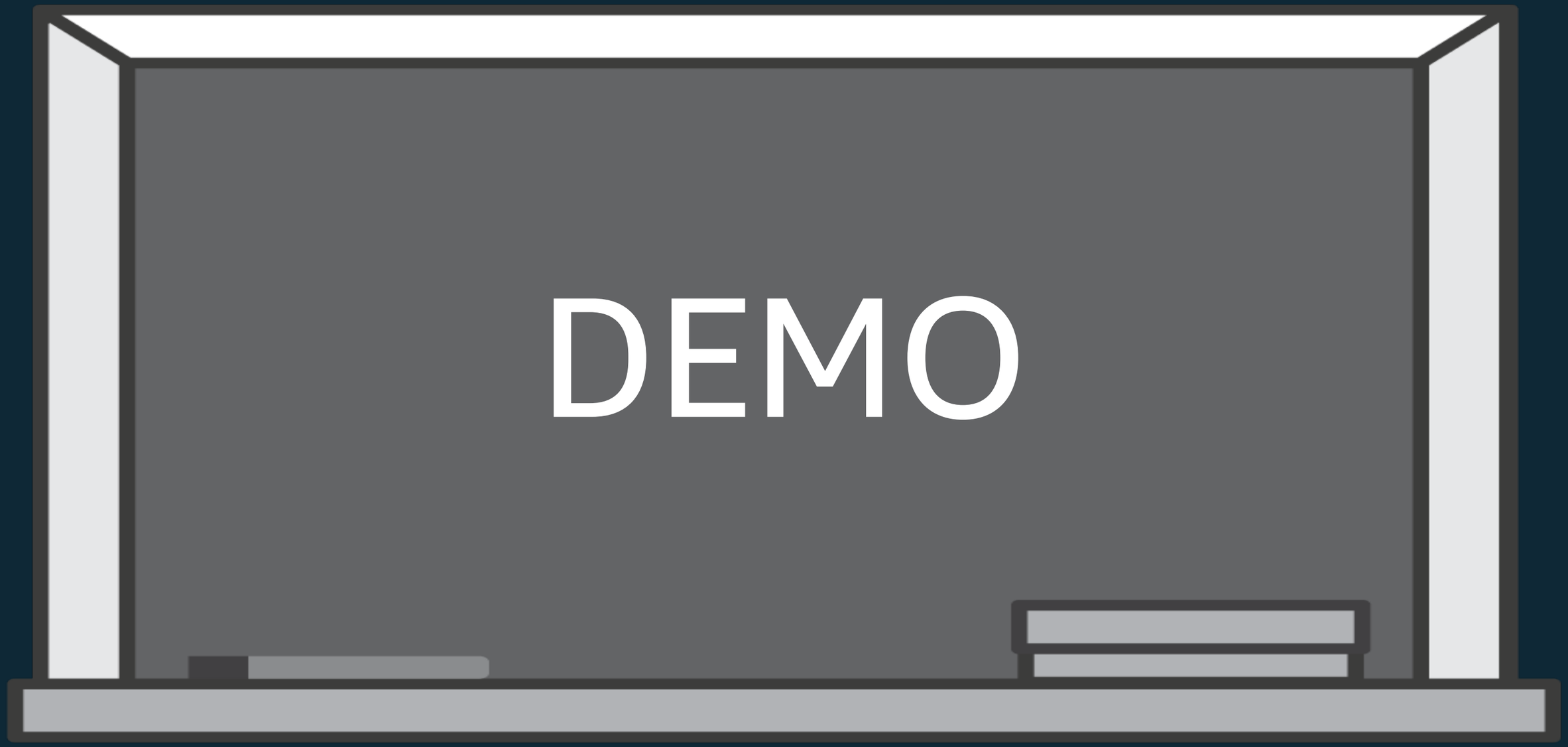
Dive deep into trace and log information

Trace Summary

Method	Response Code	Duration	Timestamp
POST	200	305ms	3 minutes (2020-05-11 15:46:53)

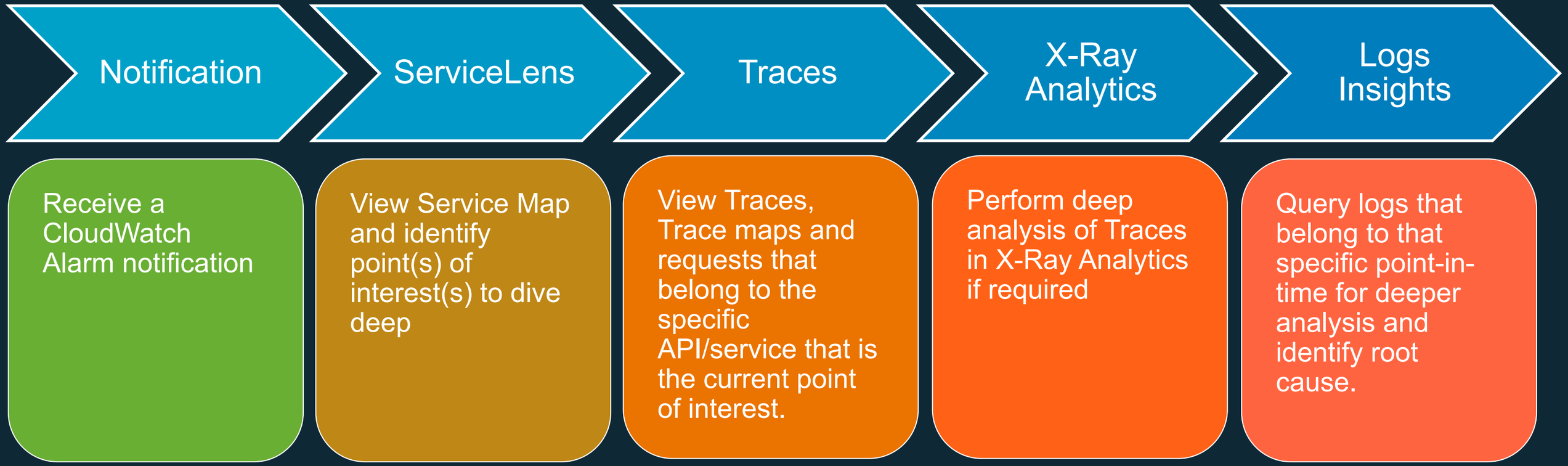
Segments Timeline Info





Bringing it all together

Troubleshooting workflow



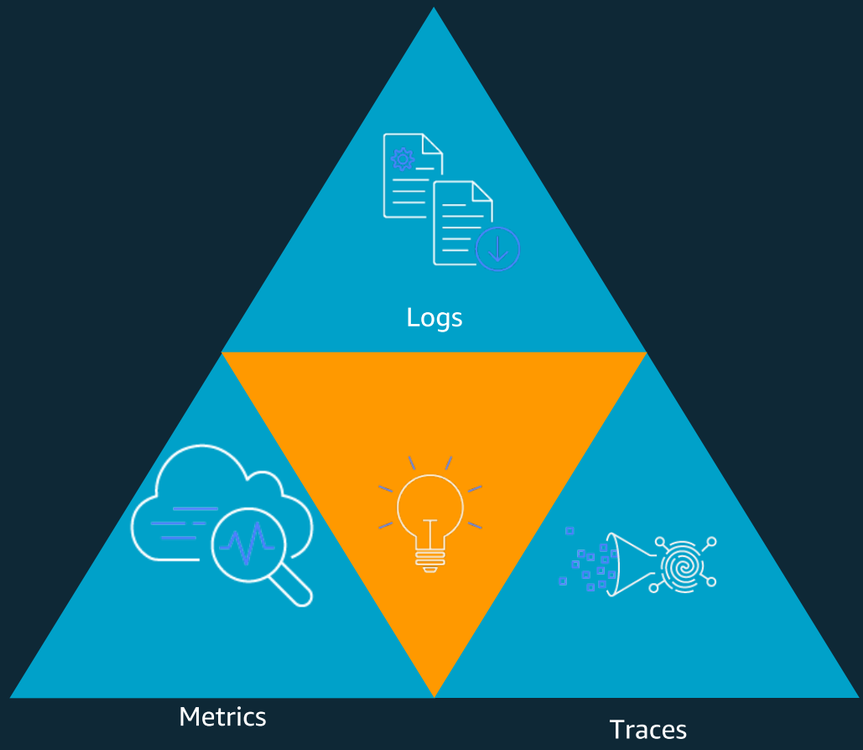
FIN/ACK (in closing)

Observability is more than just monitoring metrics:

- Data driven decision making
- Understanding states that exist between up and down.
- A culture focused on gathering as much data required to make the right decisions

Amazon CloudWatch and AWS X-Ray give you tools to gain visibility to how your systems are performing

- Embedded metrics provide a powerful 2 for 1
- ServiceLens gives you that “single-pane” view into your applications



aws.amazon.com/serverless

Serverless Computing and Applications

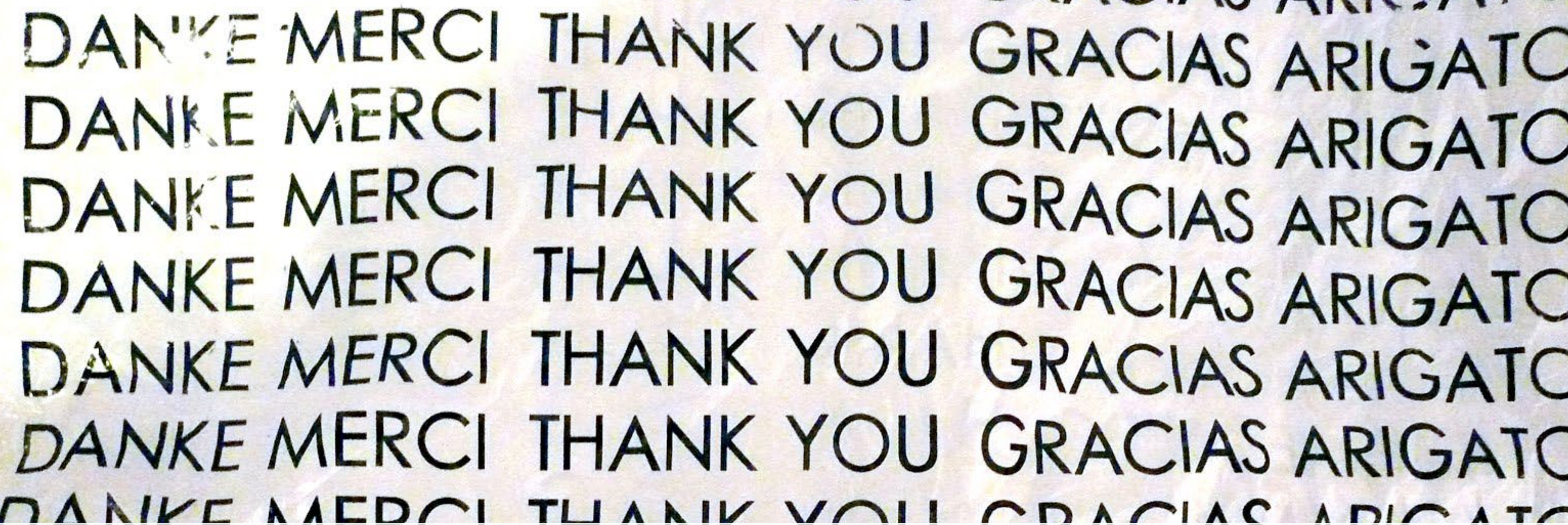
> Build and run applications without thinking about servers

Find serverless applications

Serverless computing allows you to build and run applications and services without thinking about servers. Serverless applications don't require you to provision, scale, and manage any servers. You can build them for nearly any type of application or backend service, and everything required to run and scale your application with high availability is handled for you.

Building serverless applications means that your developers can focus on their core product instead of worrying about managing and operating servers or runtimes, either in the cloud or on-premises. This reduced overhead lets developers reclaim time and energy that can be spent on developing great products which scale and that are reliable.

https://secure.flickr.com/photos/dullhunk/202872717/



Chris Munns
munns@amazon.com
@chrismunns

https://www.flickr.com/photos/theredproject/3302110152/