

Observability in Serverless Application

Ozioma Uzoegwu
Solutions Architect, AWS



What we will cover in this session

What is a serverless application?

What is observability?

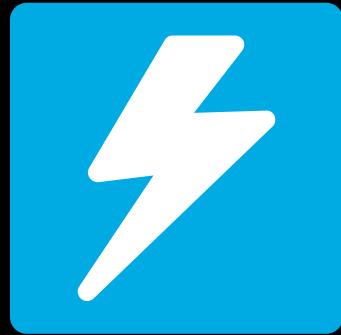
AWS services for observability

- Standard and custom metrics
- Structured logging
- Tracing

AWS open source observability services

Serverless applications

Event source



Changes in
data state



Changes in
resource state



Requests to
endpoints



Function

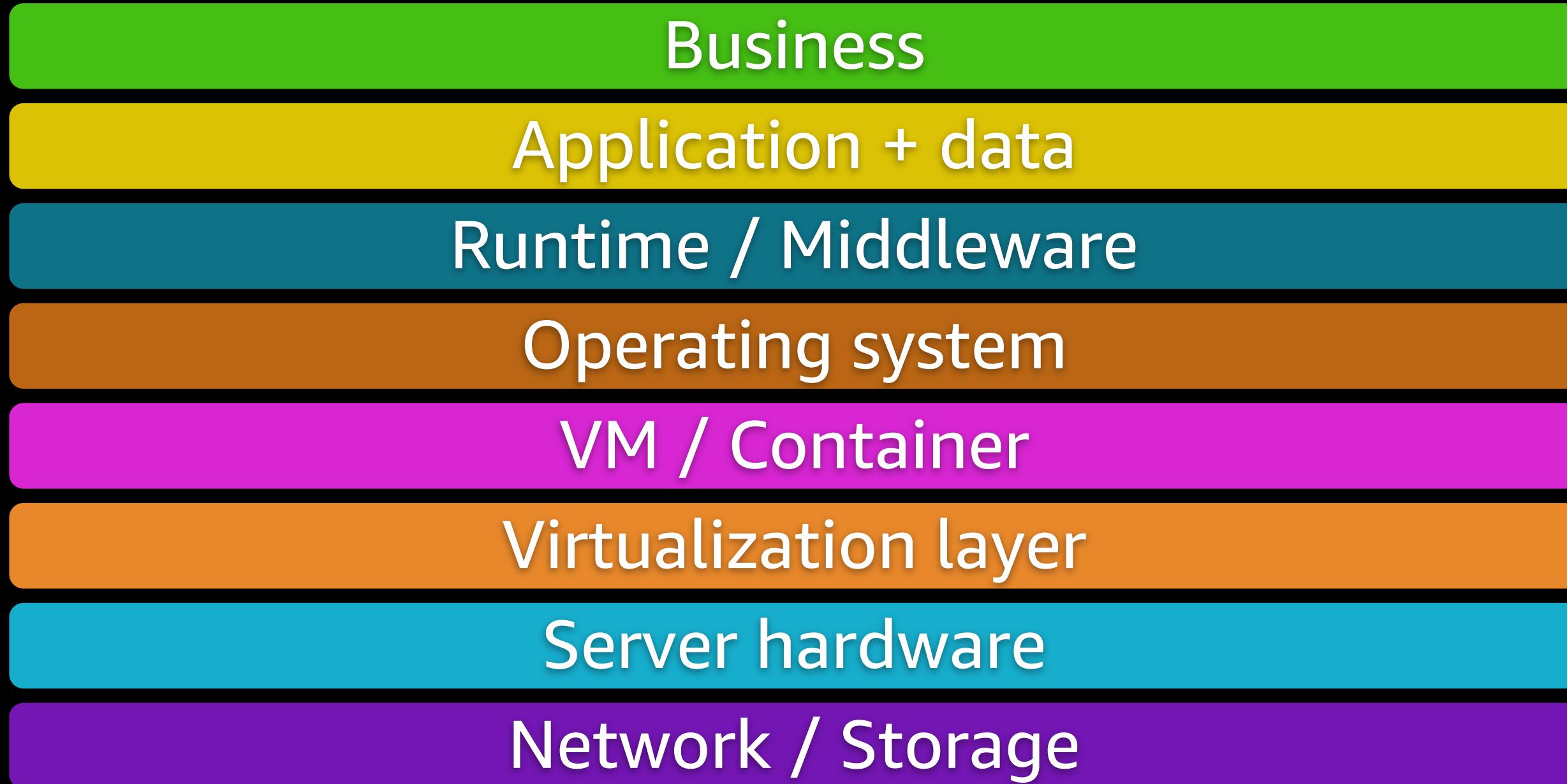


Node.js
Python
Java
Go
Ruby
.Net (C# / PowerShell)
Custom (Runtime API)

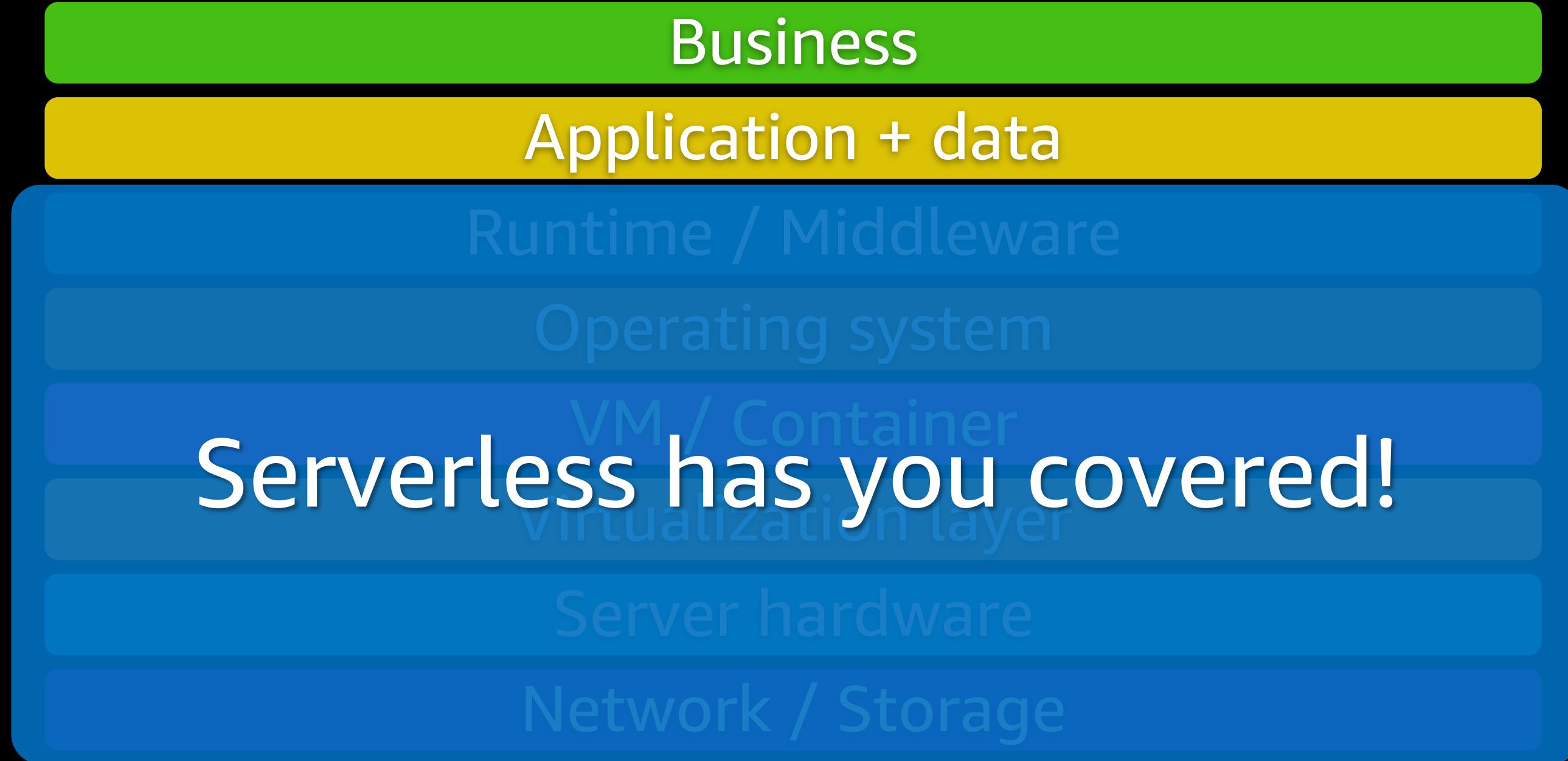
Services
(anything)



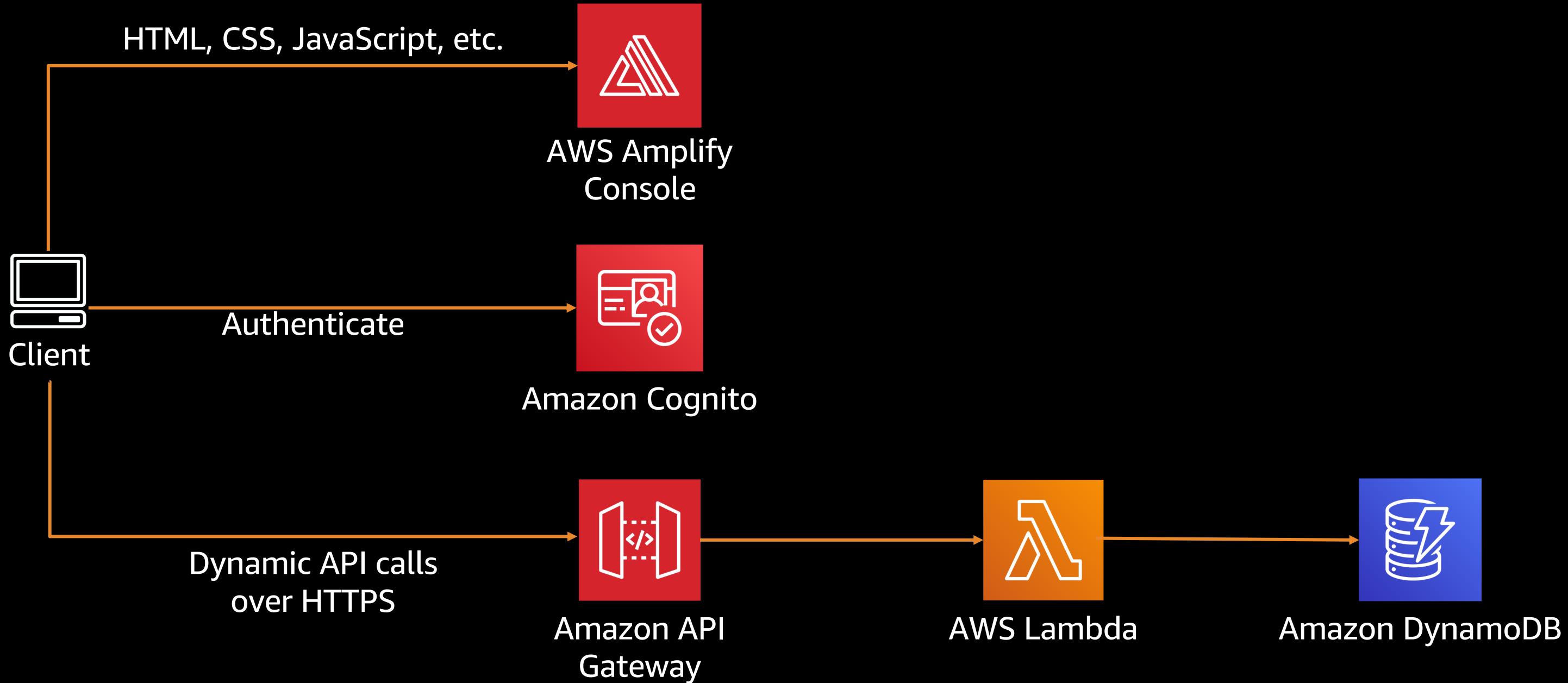
Traditional application stack



Serverless application stack



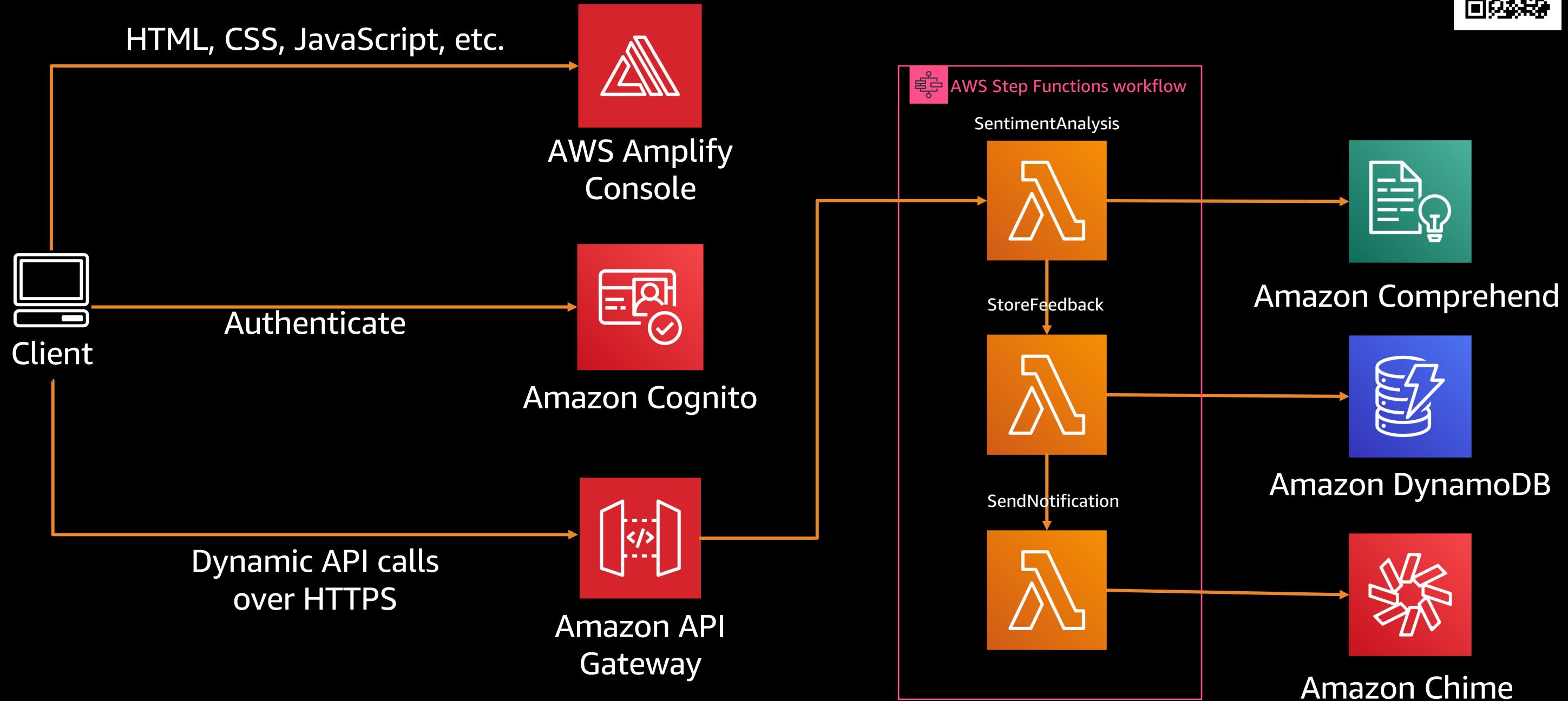
Serverless Web Application Architecture



Serverless Web Application Architecture



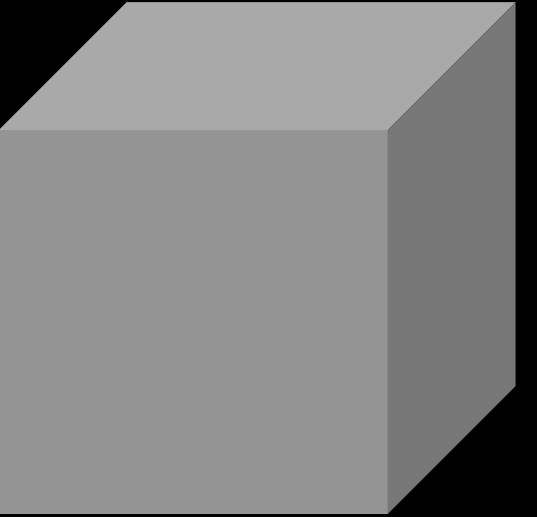
<https://s12d.com/serverless-feedback-app>



What is observability?

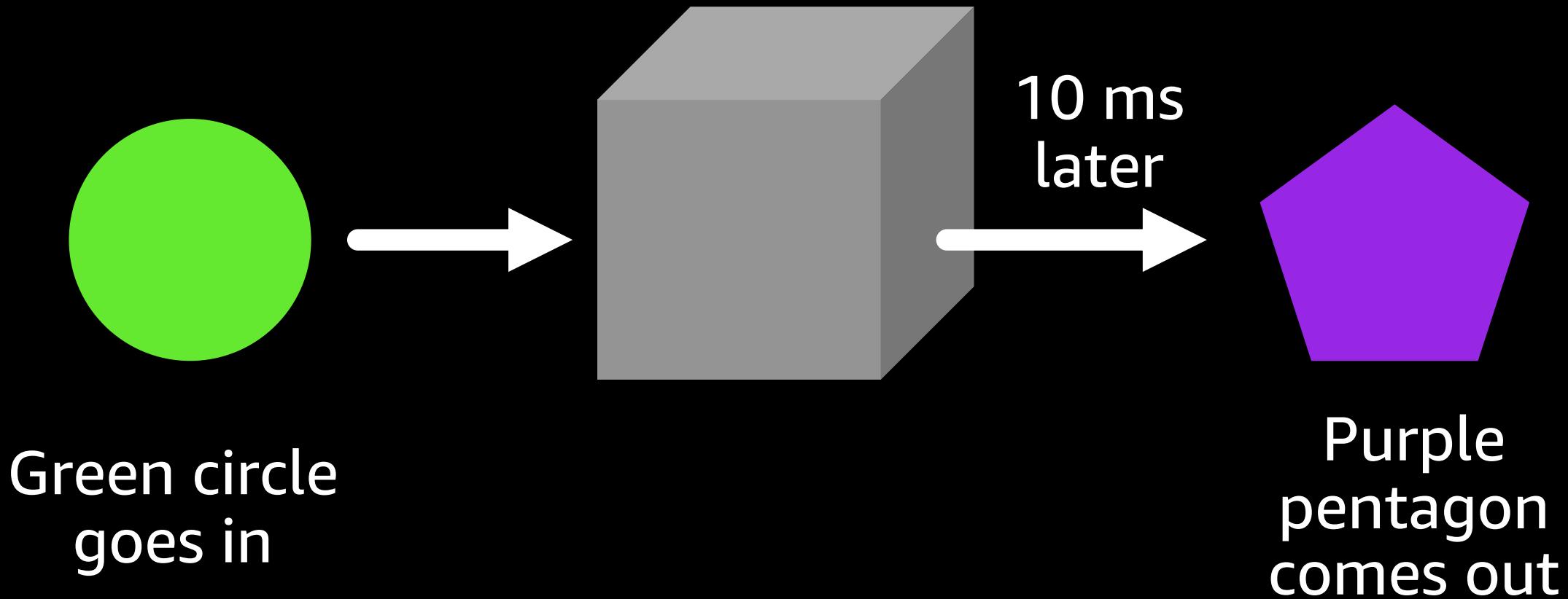
What is observability?

The "magic box"

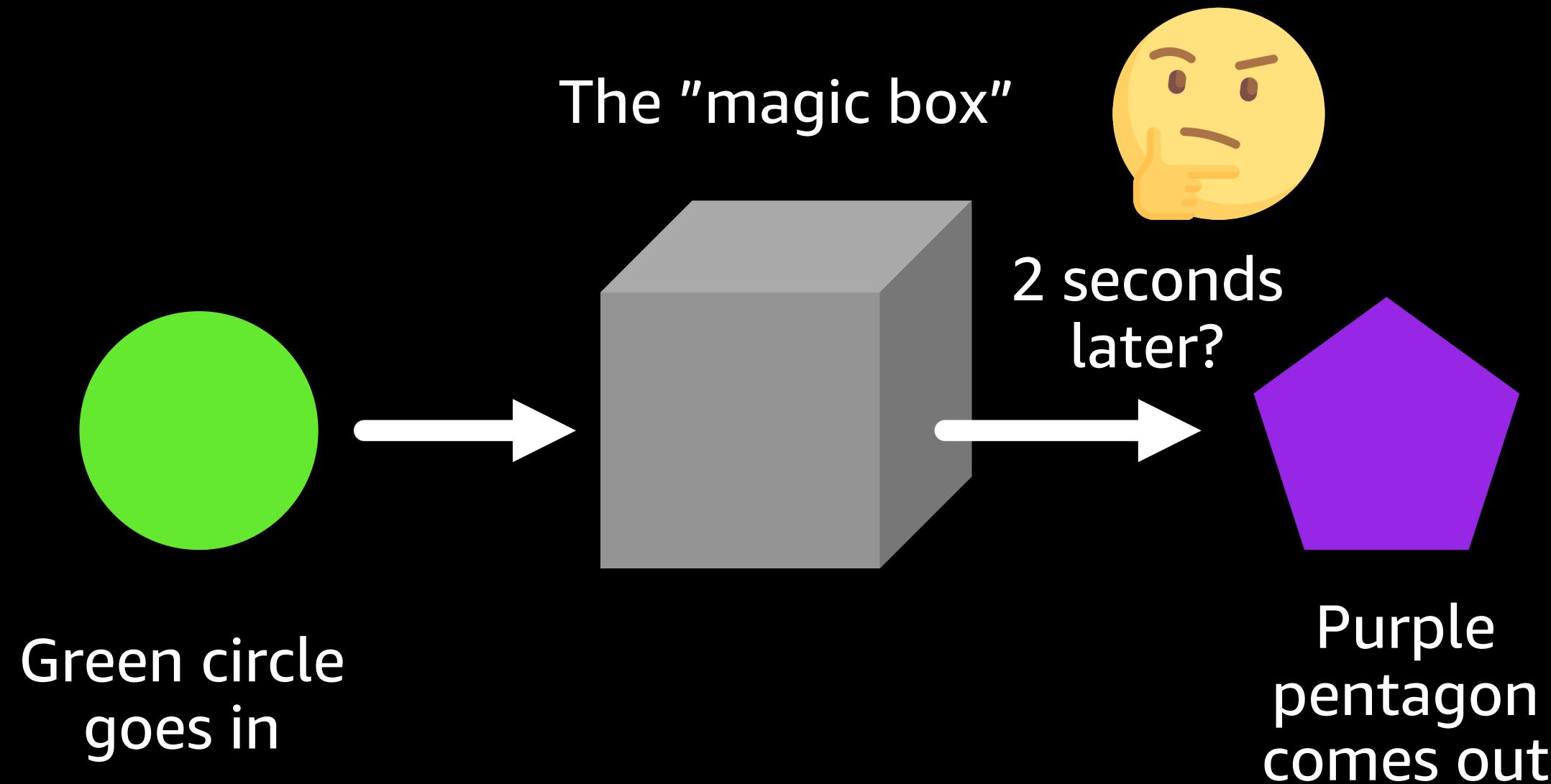


What is observability?

The "magic box"

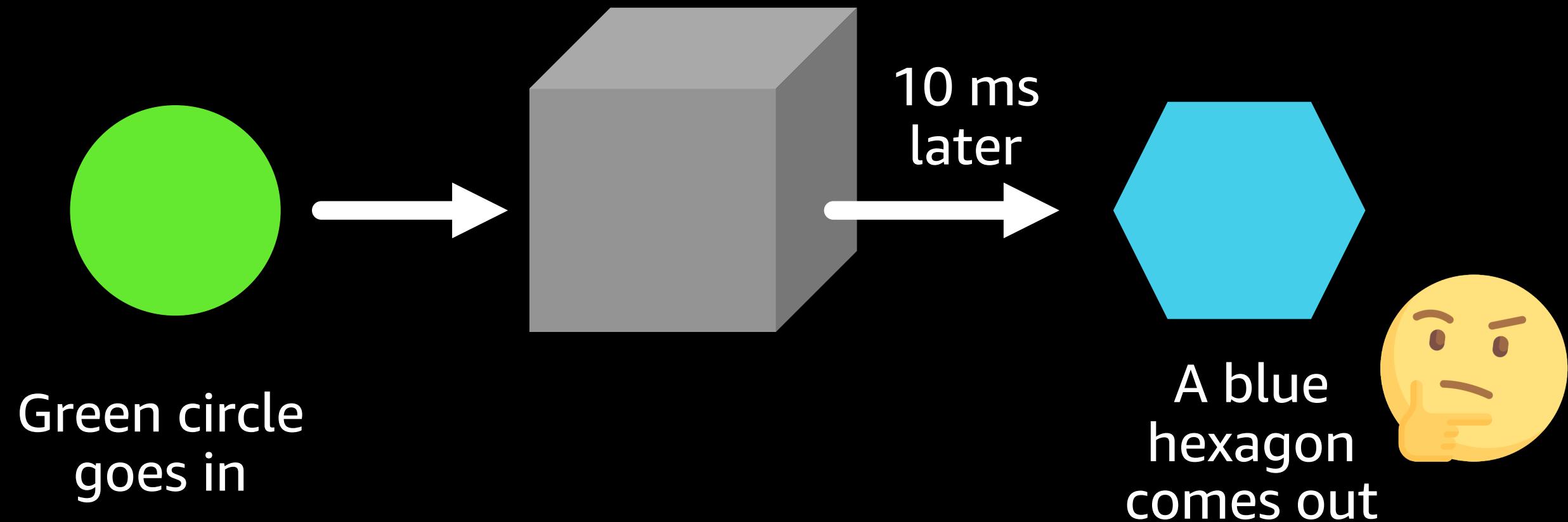


What is observability?

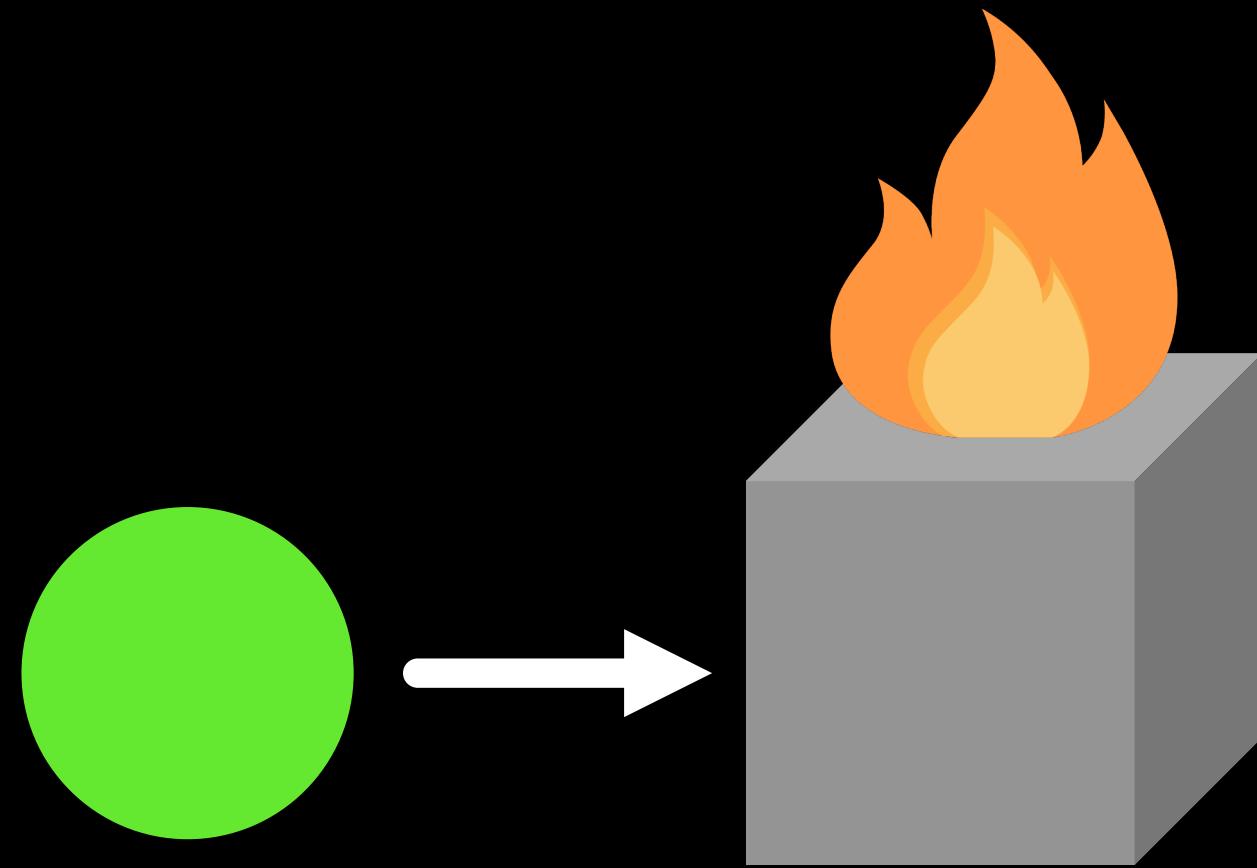


What is observability?

The "magic box"



What is observability?

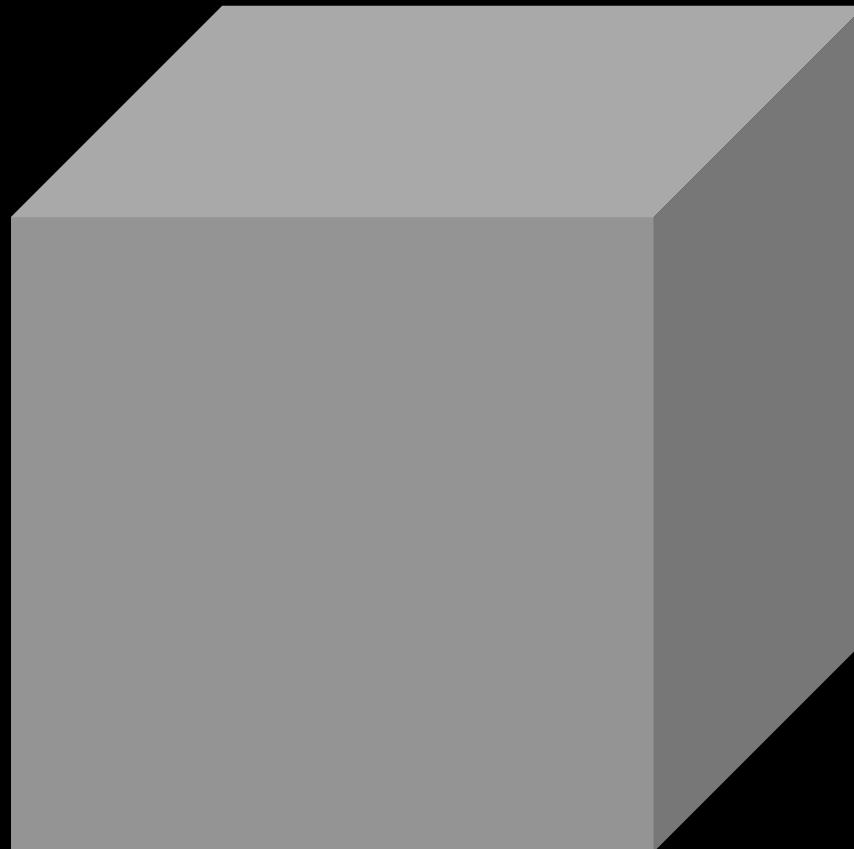


Green circle
goes in

The "magic box"
catches fire and
nothing comes
out!

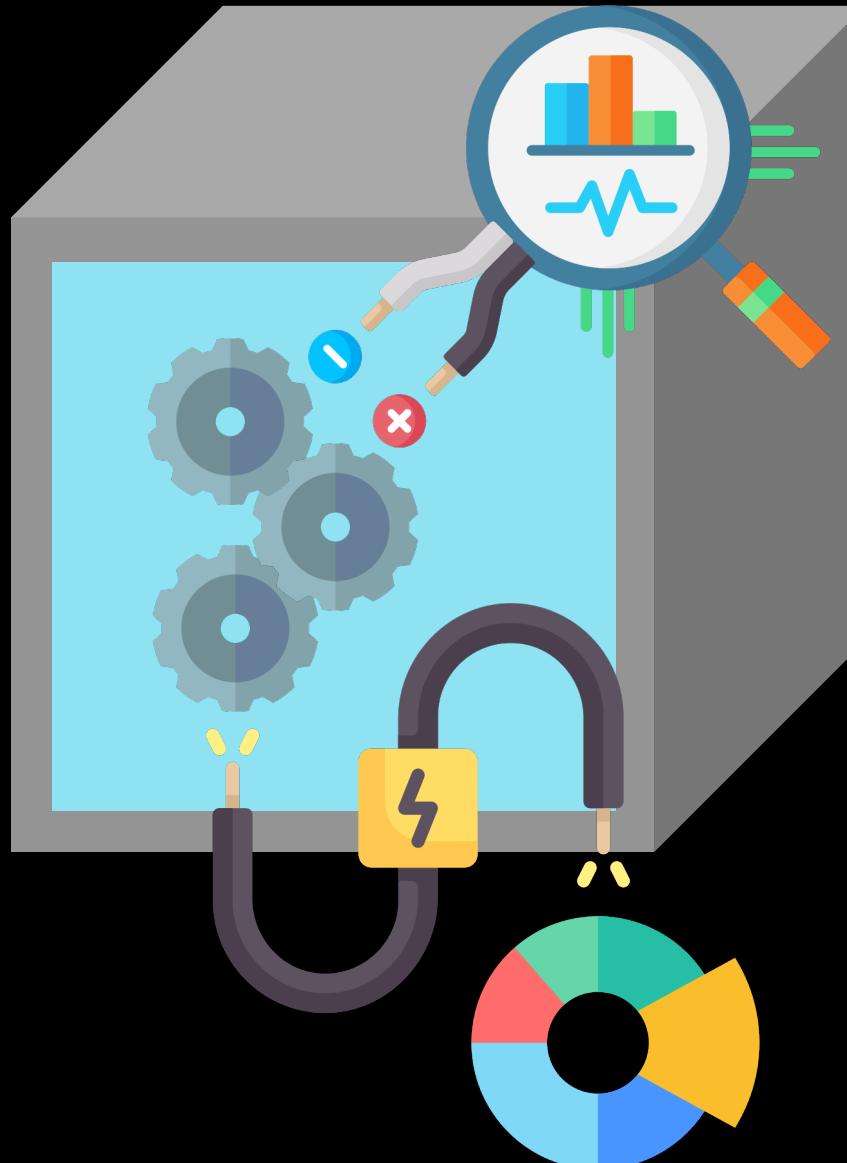


What is observability?



- We have no observability!
- What's in the box?
- Why does it behave the way it does?
- When its behavior changes, why did it change?
- What must be done to make this behavior more consistent?
- What is the usage?
- What is the business impact?

Observability must be proactive



Good observability allows you to answer questions you didn't know you needed to ask

When a problem happens you can access data about the system and understand it

Three pillars of observability tooling

Metrics

Numeric data measured at various time intervals (time series data); SLIs (request rate, error rate, duration, CPU%, etc.)

Logs

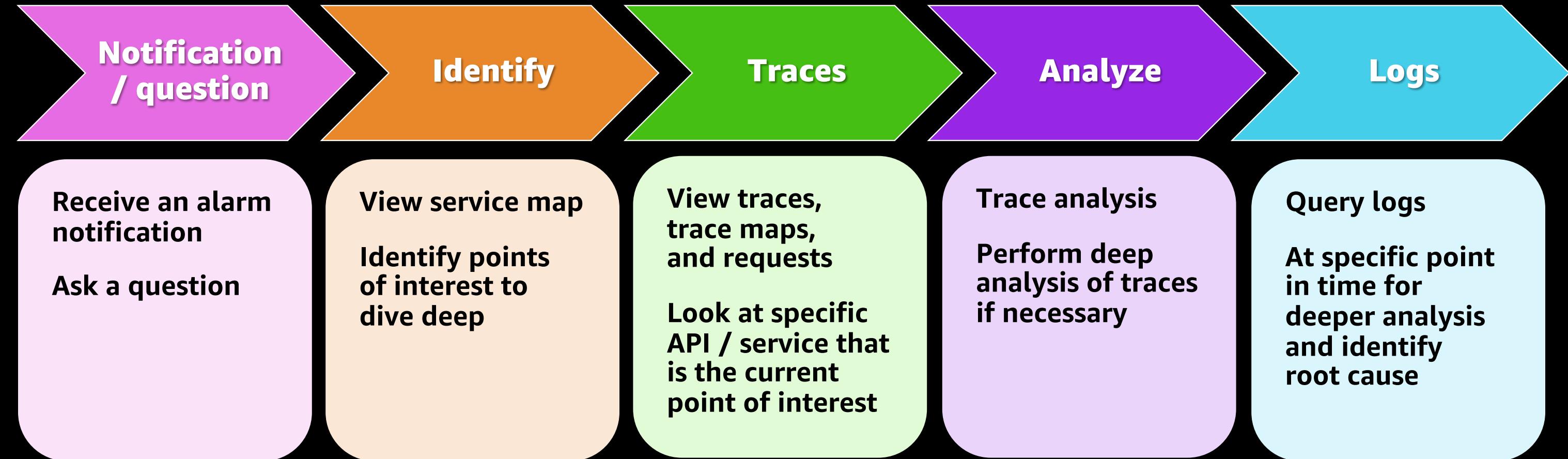
Timestamped records of discrete events that happened within an application or system, such as a failure, an error, or a state transformation

Traces

A trace represents a single user's journey across multiple applications and systems (usually microservices)

Definitions from "Distributed Systems Observability," by Cindy Sridharan. Available at:
<https://www.oreilly.com/library/view/distributed-systems-observability/9781492033431/>

Troubleshooting / query workflow



AWS services for observability

AWS services for observability



Amazon
CloudWatch



AWS X-Ray

Dashboards
Logs
Metrics
Alarms
Events

Traces
Analytics
Service map

Amazon CloudWatch

1 quadrillion+

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

Metric observations each month

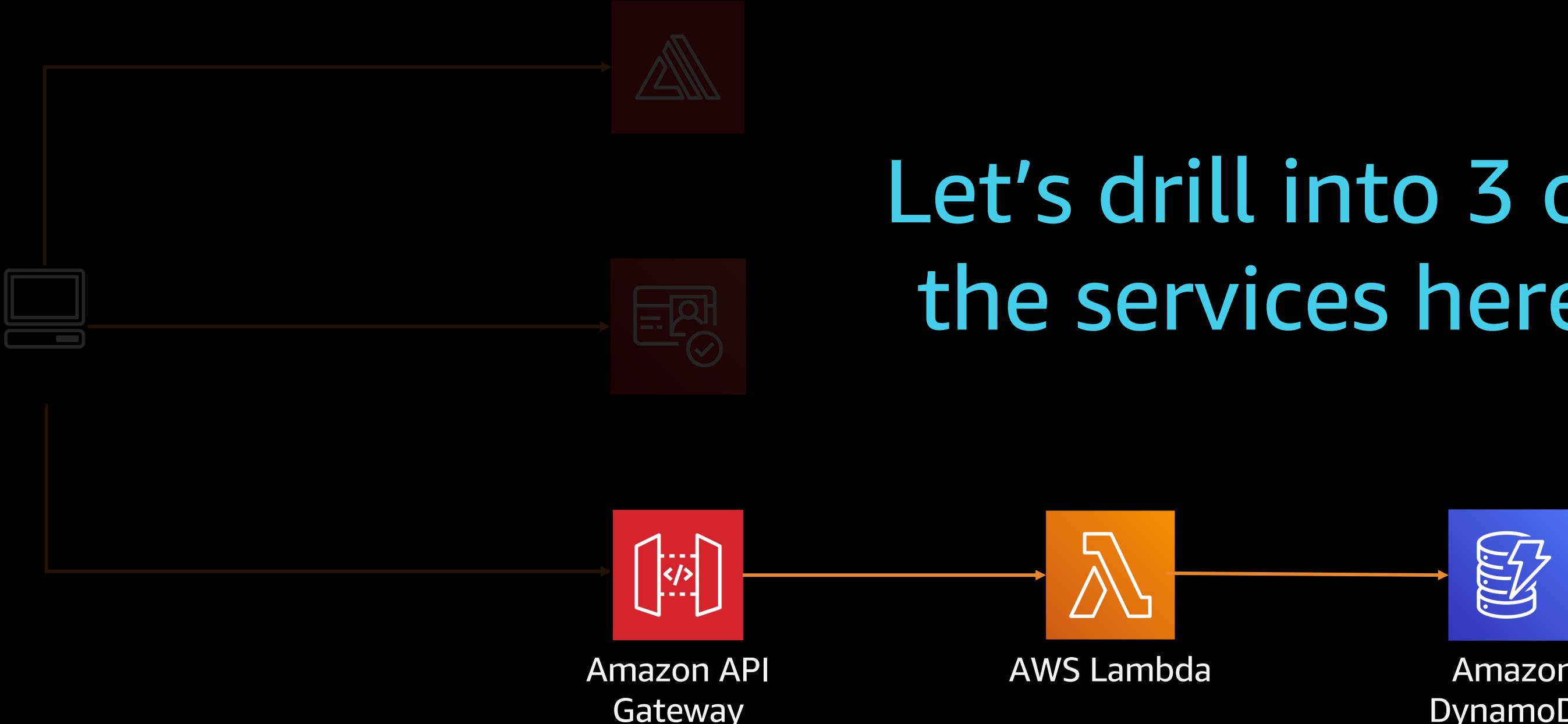
3.9 trillion

Events each month

Monitors entire
infrastructure of
AWS and
Amazon.com

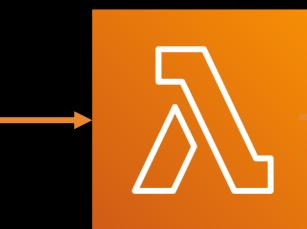
100 PB

Logs ingested each month



Let's drill into 3 of
the services here

Amazon API
Gateway



AWS Lambda



Amazon
DynamoDB

CloudWatch built-in metrics



AWS Lambda

Invocation metrics

Invocation count, **Invocation errors**, DeadLetterErrors, DestinationDeliveryFailures, **Throttles**, ProvisionedConcurrencyInvocations, ProvisionedConcurrencySpilloverInvocations

Performance metrics

Duration, IteratorAge

Concurrency metrics

ConcurrentExecutions, ProvisionedConcurrentExecutions, ProvisionedConcurrencyUtilization, UnreservedConcurrentExecutions

Amazon API Gateway

REST APIs

API request count, **Latency**, 4XXs, **5XXs**, **IntegrationLatency**, CacheHitCount, CacheMissCount

HTTP APIs

API request count, **Latency**, 4XXs, **5XXs**, **IntegrationLatency**, DataProcessed

WebSocket APIs

ConnectCount, MessageCount, **IntegrationError**, ClientError, ExecutionError, **IntegrationLatency**

CloudWatch built-in metrics



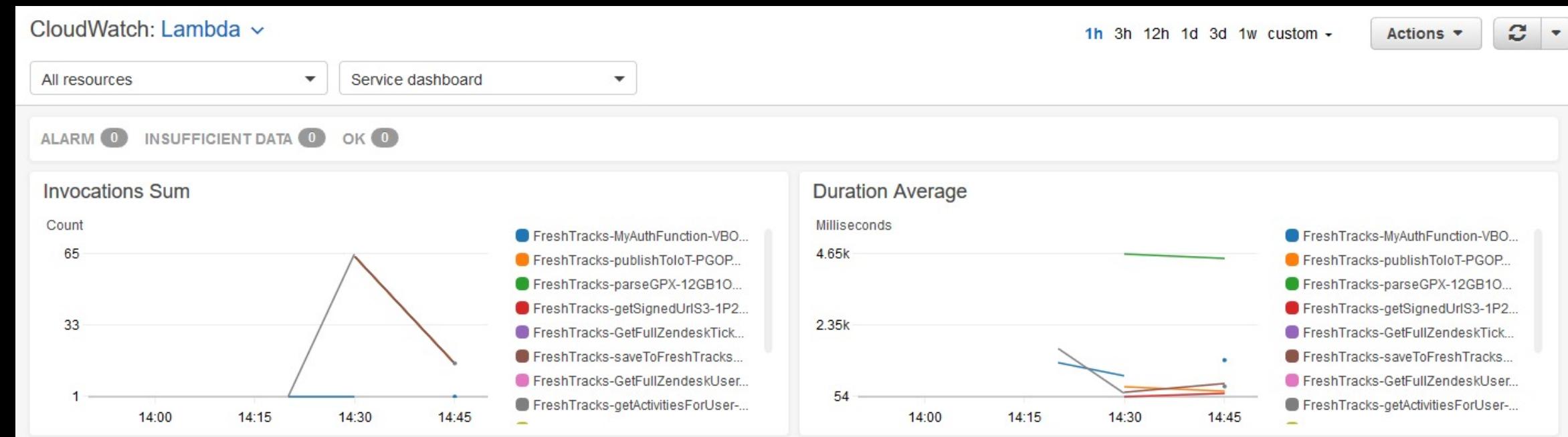
Amazon DynamoDB

AccountMaxReads,
AccountMaxTableLevelReads,
AccountMaxTableLevelWrites,
AccountMaxWrites,
AccountProvisionedReadCapacityUtilization,
AccountProvisionedWriteCapacityUtilization,
ConditionalCheckFailedRequests,
ConsumedReadCapacityUnits,
ConsumedWriteCapacityUnits,
MaxProvisionedTableReadCapacityUtilization,
MaxProvisionedTableWriteCapacityUtilization,
OnlineIndexConsumedWriteCapacity,

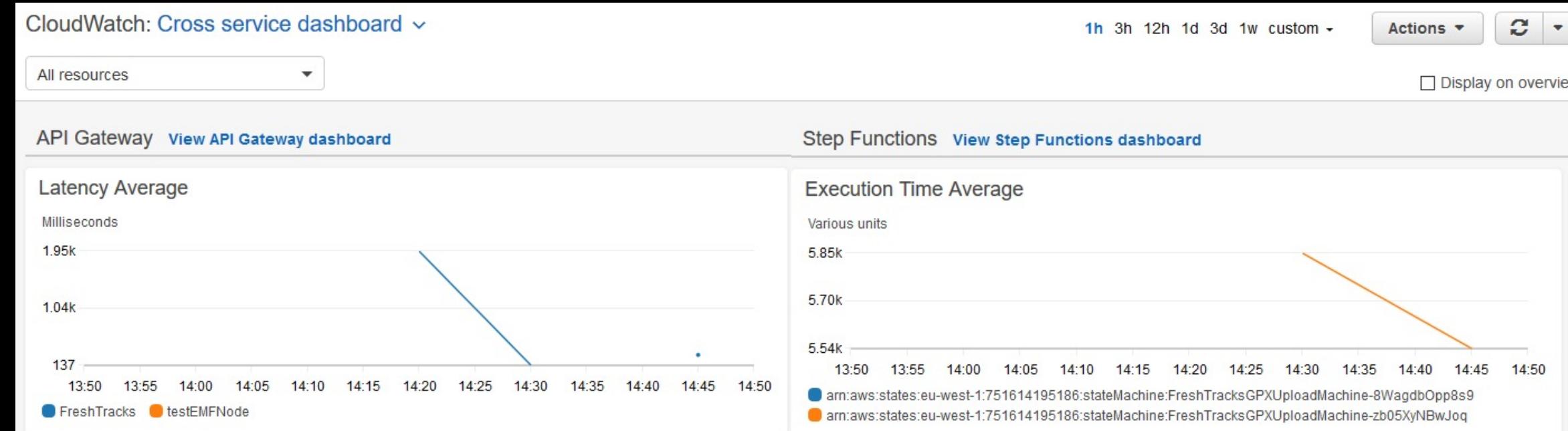
OnlineIndexPercentageProgress,
OnlineIndexThrottleEvents,
PendingReplicationCount,
ProvisionedReadCapacityUnits,
ProvisionedWriteCapacityUnits,
ReadThrottleEvents, ReplicationLatency,
ReturnedBytes, ReturnedItemCount,
ReturnedRecordsCount,
SuccessfulRequestLatency, **SystemErrors**,
TimeToLiveDeletedItemCount,
ThrottledRequests, TransactionConflict,
UserErrors, **WriteThrottleEvents**

Per-service and cross-service dashboards

Per-service
metrics
dashboard



Cross-service
metrics
dashboard



Built-in metrics often not enough

What about business / customer metrics?

Measure application performance against business goals

Revenue, signups, pictures uploaded, perceived latency, page views, etc.

How operationally stable is the application?

Continuous integration / deployment feedback time, mean time between failure and recovery, number of on-call pages and time to resolution, etc.

What about caught errors, warnings?

Caught exceptions are not counted as errors on AWS Lambda

What if I want to use other dimensions?

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

Creating custom metrics

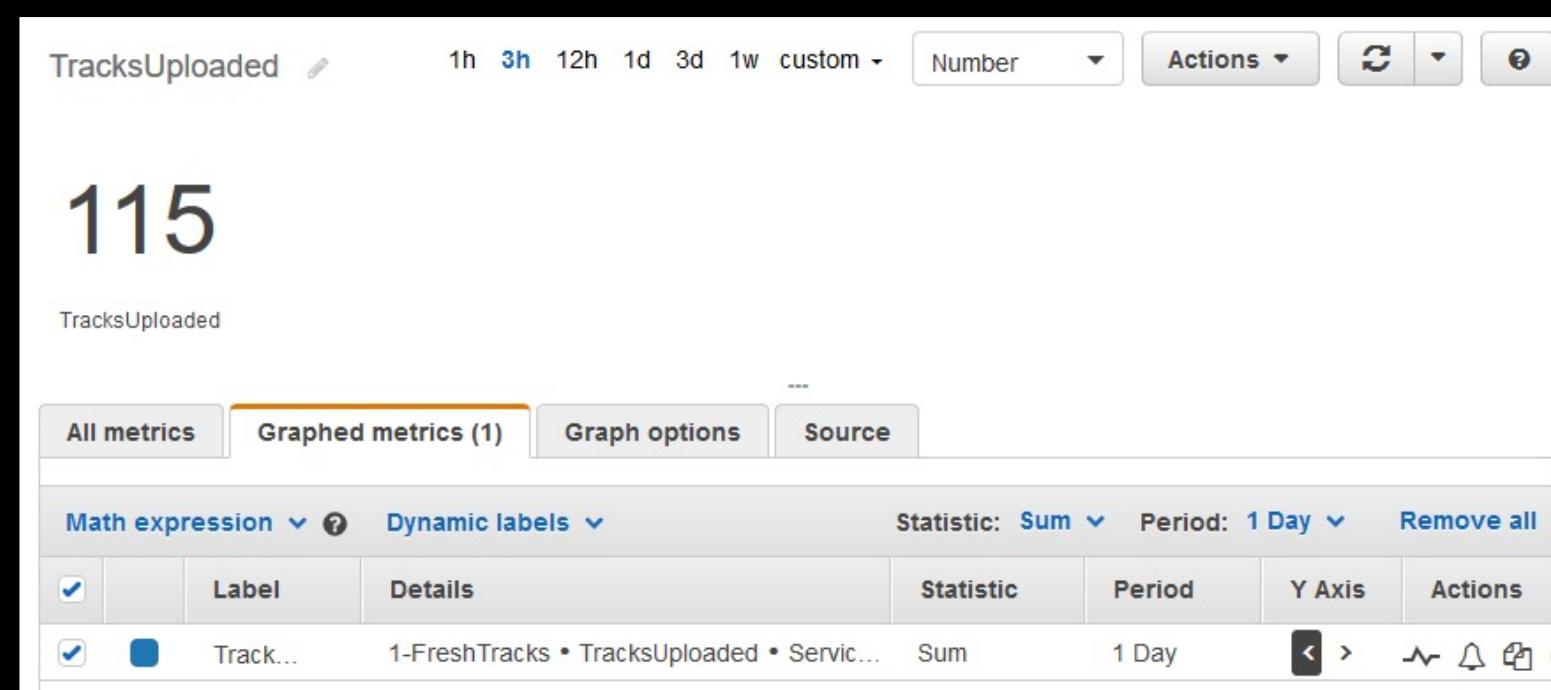
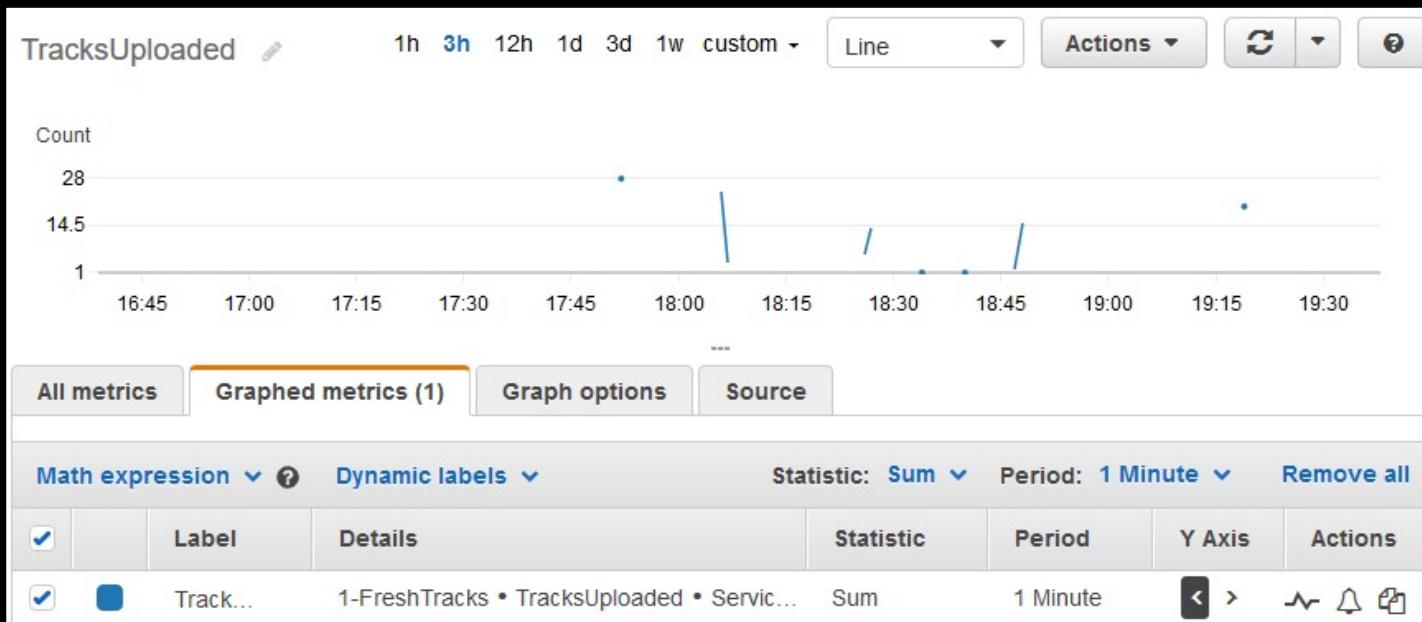
Useful for application, business, and operations metrics

- 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

```
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())  
}
```

Improved by embedded metric format (covering shortly)

Visualize with CloudWatch metrics graphs



The CloudWatch Metrics Explorer interface shows metrics for Lambda functions. It includes a sidebar for 'Metrics Info' with a search bar and a dropdown for 'Lambda by runtime'. Below this, specific metrics are listed: 'Lambda Function: Duration: p90' and 'Lambda Function: Errors: Sum'. There is also a link to 'Show more chosen options (+2)' and a 'Clear all' button.

CloudWatch Metrics Explorer
<https://s12d.com/cw-me>



CloudWatch logging

Built-in logging

API Gateway logging

- REST: Two levels of logging, ERROR and INFO
 - Set globally in stage, or override per method
 - Optionally log method request / body content
- HTTP APIs and WebSocket APIs with logging variables

Lambda logging

- Logging directly from your code with your language's equivalent of `console.log()` – basic request information included
- JSON structured logging via `PutMetricData` API or embedded metric format, which includes invocation information

Export logs to Amazon OpenSearch or Amazon S3

- Explore with Kibana or Amazon Athena / Amazon QuickSight



CloudWatch embedded metric format

AUTOMATICALLY GENERATE METRICS FROM STRUCTURED CLOUDWATCH LOGS

Embed custom metrics alongside detailed log event data

Can send structured format
in `PutLogEvents` API call
with specific format

Asynchronous

Open-source client libraries
available for

- Node.js
- Python
- Java



<https://s12d.com/cwl-emf-client>

Installation

```
npm install aws-embedded-metrics
```

Usage

To get a metric logger, you can either

Using the metricScope decorator with

```
const { metricScope, Unit } = require('aws-embedded-metrics');

const myFunc = metricScope(metrics =>
  async () => {
    metrics.putDimensions({ Service: "MyService" });
    metrics.putMetric("ProcessingLatency", 123);
    metrics.setProperty("RequestId", "422b1");
    // ...
  });
  await myFunc();
```

Installation

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

Usage

To get a metric logger, you can do

```
from aws_embedded_metrics import MetricsLogger

@metric_scope
def my_handler(metrics):
    metrics.put_dimensions({"Service": "MyService"})
    metrics.put_metric("ProcessingLatency", 123)
    metrics.set_property("RequestId", "422b1")
    metrics.set_property("DevEnv", "true")

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

Usage

To use a metric logger, you need to manually create and flush the

```
import software.amazon.cloudwatchlogs.emf.logger.MetricsLogger
import software.amazon.cloudwatchlogs.emf.model.DimensionSet
import software.amazon.cloudwatchlogs.emf.model.Metric
import software.amazon.cloudwatchlogs.emf.model.Property

class Example {
    public static void main(String[] args) {
        MetricsLogger logger = new MetricsLogger();
        logger.putDimensions(DimensionSet.of("Service", "MyService"));
        logger.putMetric("ProcessingLatency", 123);
        logger.setProperty("RequestId", "422b1");
        logger.flush();
    }
}
```

CloudWatch 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": {  
        "functionVersion": "$LATEST",  
        "Timestamp": 1576064416496,  
        "CloudWatchMetrics": [  
            {"Namespace": "ecommerce-cart",  
             "Dimensions": [  
                 {"Environment": "Category", "Category": "Category"},  
                 {"Environment": "Region", "Region": "Region"}],  
             "Metrics": [  
                 {"Name": "PriceInCart", "Unit": "None"},  
                 {"Name": "QuantityInCart", "Unit": "None"}]}],  
        }]  
    }  
}
```

Amazon CloudWatch Logs Insights

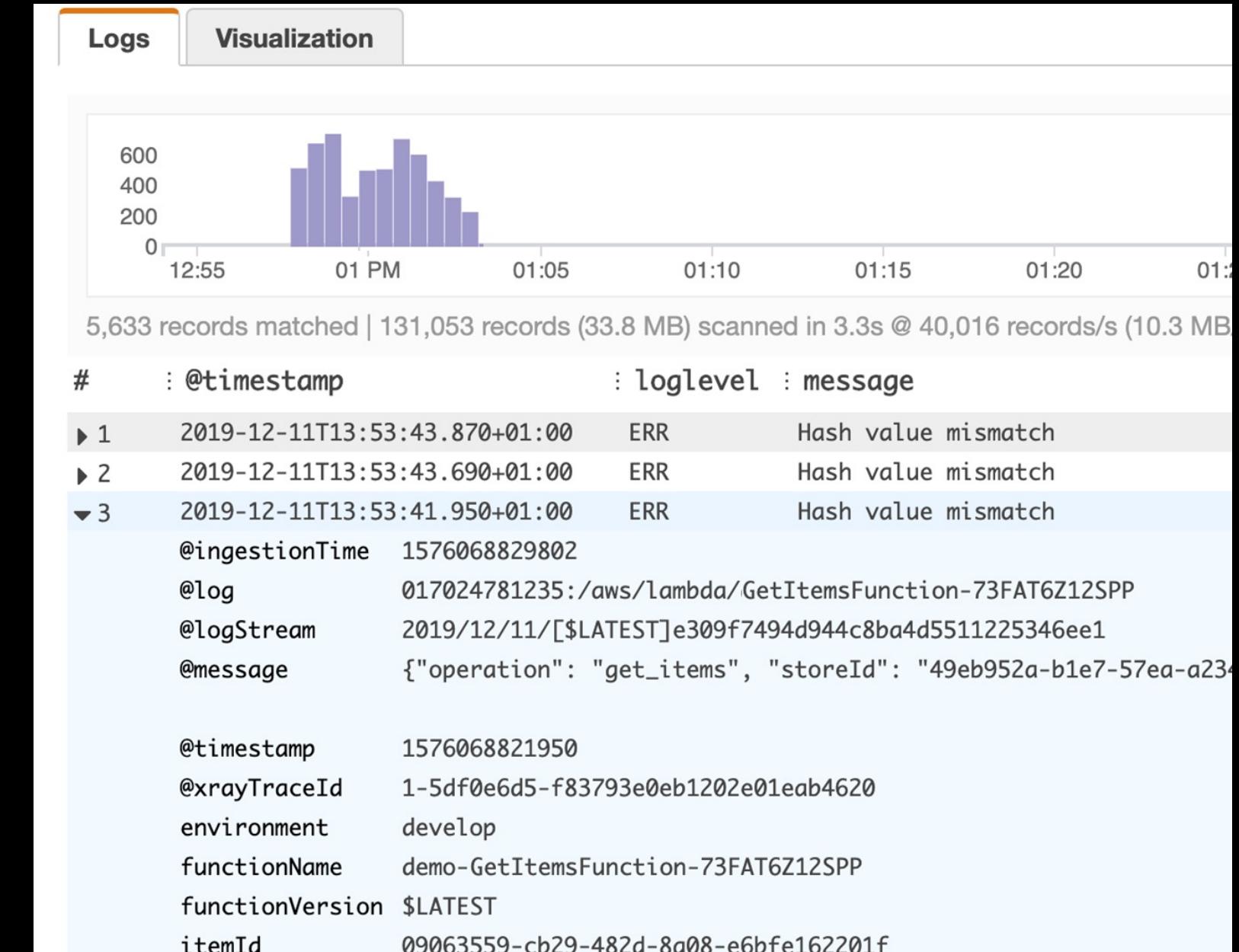
Amazon CloudWatch Logs Insights

Interactively search and analyze your log data in Amazon CloudWatch Logs

Drive actionable intelligence to address operational issues without needing to provision servers or manage software

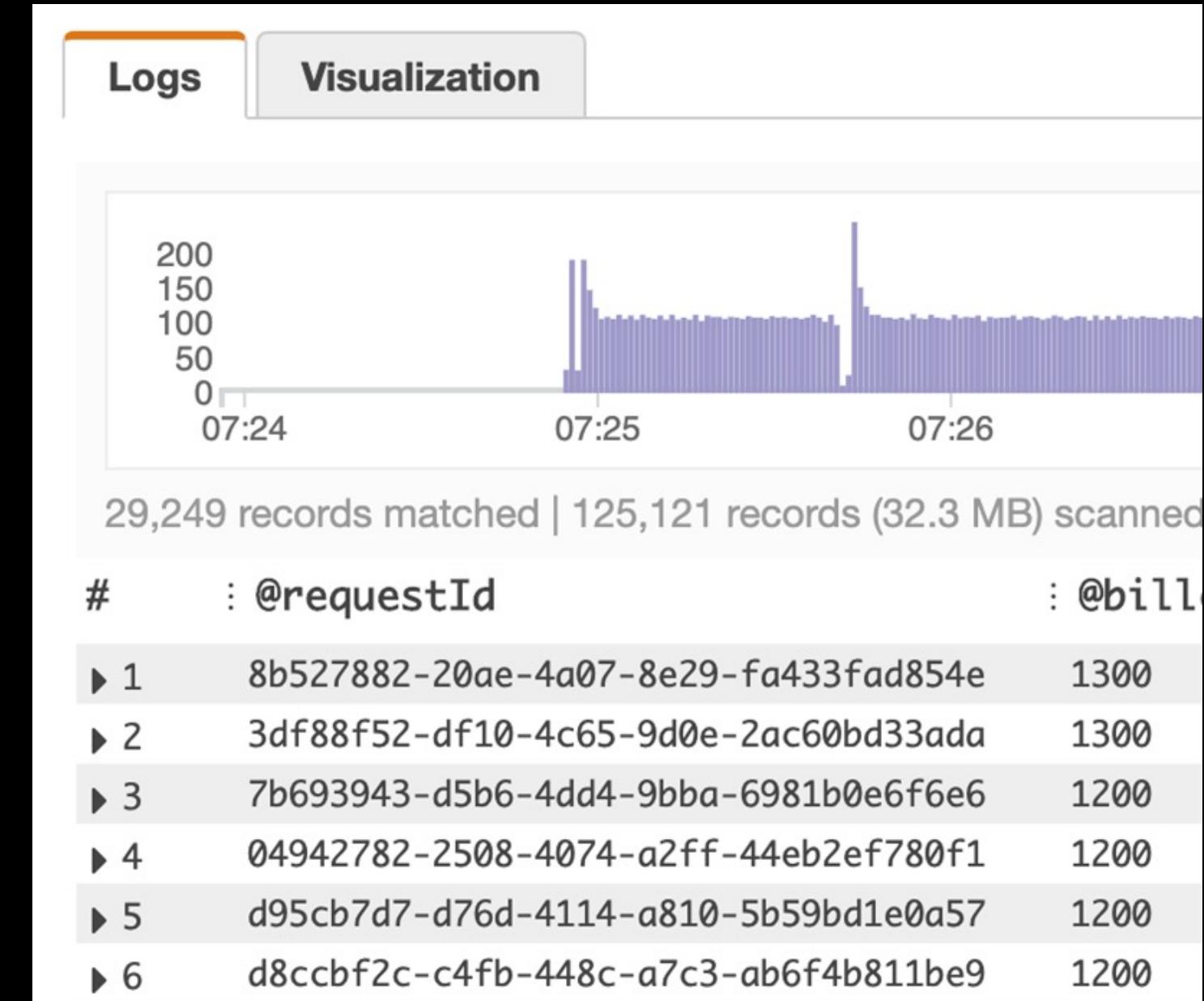
- Processes structured log data
- Flexible purpose-built query language
- Query up to 20 log groups
- Save queries

```
fields @timestamp, @logLevel, @message  
| filter @logLevel == "ERR"  
| sort @timestamp desc  
| limit 100
```



Top 100 most expensive invocations

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



Checking Lambda performance

P90 latency, total invokes, and max latency for each 5-minute window

```
filter @type =  
"REPORT"  
| stats  
avg(@duration),  
max(@duration),  
min(@duration),  
pct(@duration,  
90),  
count(@duration)  
by bin(5m)
```

Logs Visualization Export results ▾ Add to dashboard

Showing 48 of 2,352 records matched ⓘ Show histogram

14,276 records (4.8 MB) scanned in 3.8s @ 3,745 records/s (1.3 MB/s)

#	bin(5m)	avg(@duration)	max(@duration)	min(@duration)	pct(@duration, 90)	count(@duration)
► 1	2020-10-21T22:00:	2665.7931	5299.88	241.51	4454.03	48
► 2	2020-10-21T21:45:	2711.1933	4923.12	237.99	4477.22	48
► 3	2020-10-21T21:30:	2567.0235	5169.22	304.31	4253.03	48
► 4	2020-10-21T21:15:	2671.319	5105.17	338.53	4199.89	48
► 5	2020-10-21T21:00:	2528.9225	4577.06	231.42	4114	48
► 6	2020-10-21T20:45:	2683.5394	5182.84	319.67	4127.31	48
► 7	2020-10-21T20:30:	2443.1156	5007.75	250.96	3692.27	48
► 8	2020-10-21T20:15:	2627.3131	5049.13	304.19	4218.91	48
► 9	2020-10-21T20:00:	2705.4185	5044.18	251.02	4620.97	48
► 10	2020-10-21T19:45:	2693.3875	5037.74	270.23	4496.73	48
► 11	2020-10-21T19:30:	2692.1761	13310.84	255.78	4237.32	96

Creating CloudWatch alarms

The screenshot shows two side-by-side interfaces for creating CloudWatch alarms.

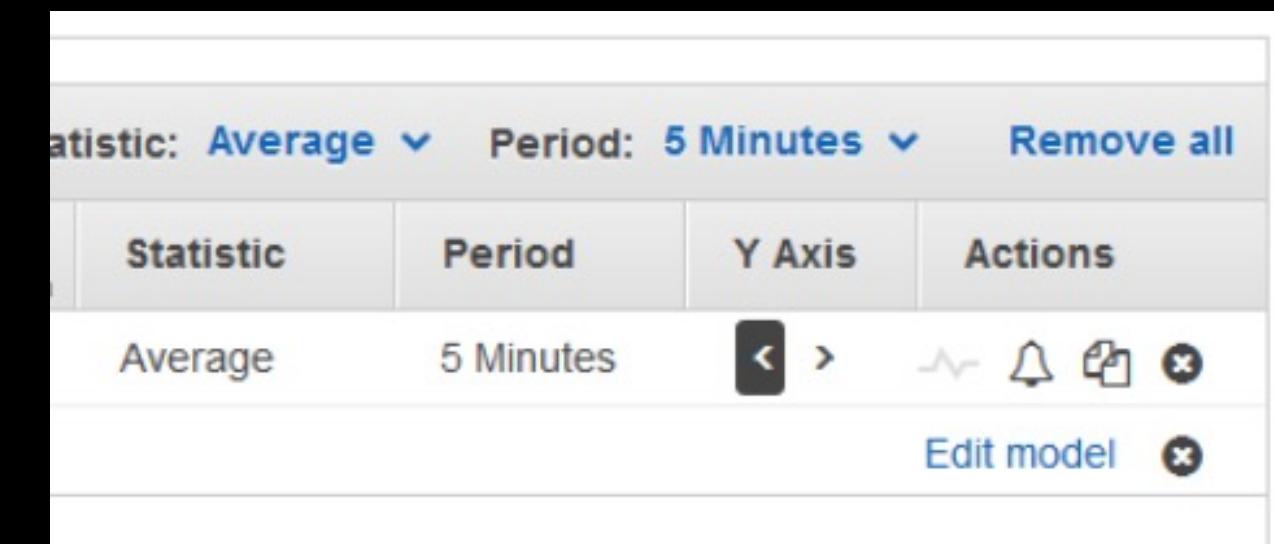
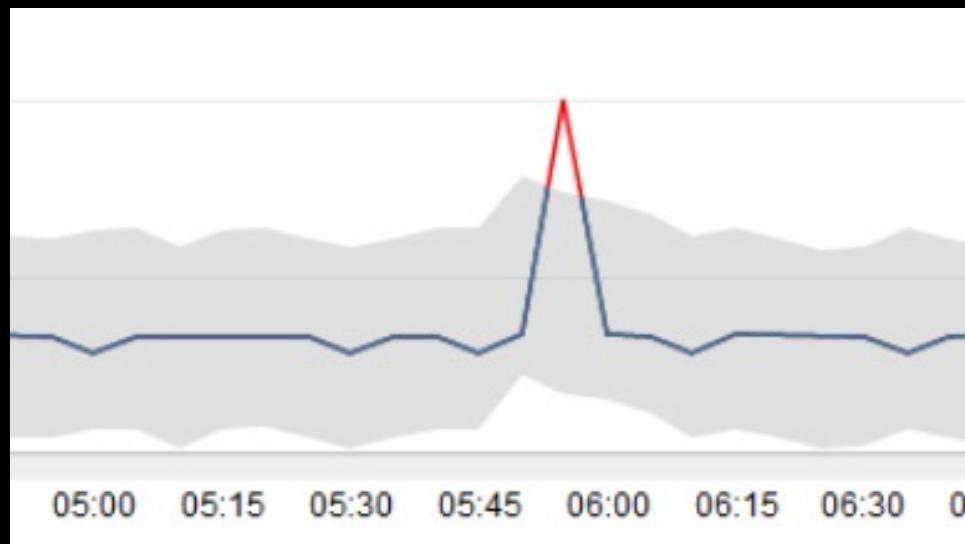
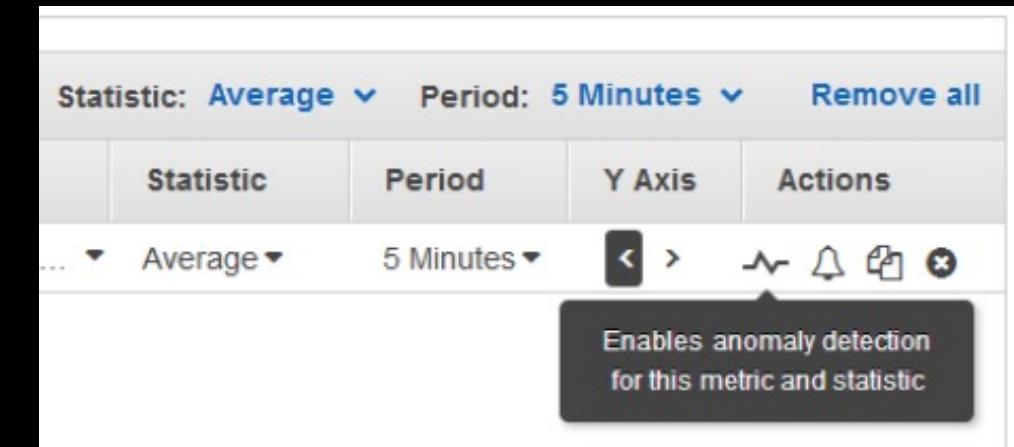
Left Panel (Metrics Explorer):

- Metric Selection:** Shows a list of metrics under "LogGroup (14)" and "Metric Name". The "TracksUploaded" metric is selected and highlighted with a green box.
- Graph View:** A line graph showing the count of "TracksUploaded" over time. The Y-axis ranges from 0 to 60, and the X-axis shows hours from 12:00 to 15:00. A red threshold line is drawn at a value of 5. The graph indicates that the metric crosses below this threshold around 14:30.
- Configuration Fields:**
 - Statistic:** Set to "Sum".
 - Period:** Set to "5 minutes".

Right Panel (CloudWatch Alarms Creation):

- Conditions:**
 - Threshold type:** "Static" is selected (highlighted with a green box). Subtext: "Use a value as a threshold".
 - Comparison Operator:** "Lower < threshold" is selected (highlighted with a green box).
 - Value:** The threshold value is set to "5" (highlighted with a green box). Subtext: "Must be a number".
- Notification:**
 - Alarm state trigger:** "In alarm" is selected (highlighted with a green box). Subtext: "The metric or expression is outside of the defined threshold."
 - OK:** "OK" is defined as "The metric or expression is within the defined threshold."
 - Insufficient data:** "Insufficient data" is defined as "The alarm has just started or not enough data is available."
- Select an SNS topic:** "Select an existing SNS topic" is selected (highlighted with a green box).
 - Topic Selection:** "FT_CWL_Topic" is selected in the dropdown.
 - Email (endpoints):** An email address is listed: "████████@amazon.com - View in SNS Console".
 - Add notification:** A button to add more notifications.

Using CloudWatch anomaly detection alarms





AWS X-Ray

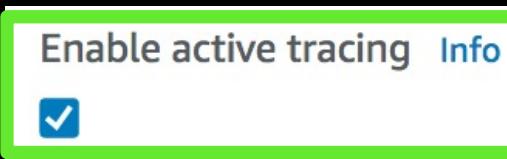
End-to-end view of requests flowing through an application

- Lambda – instruments incoming requests for all supported languages and can capture calls made in code



Globals:
Function:
Tracing: Active

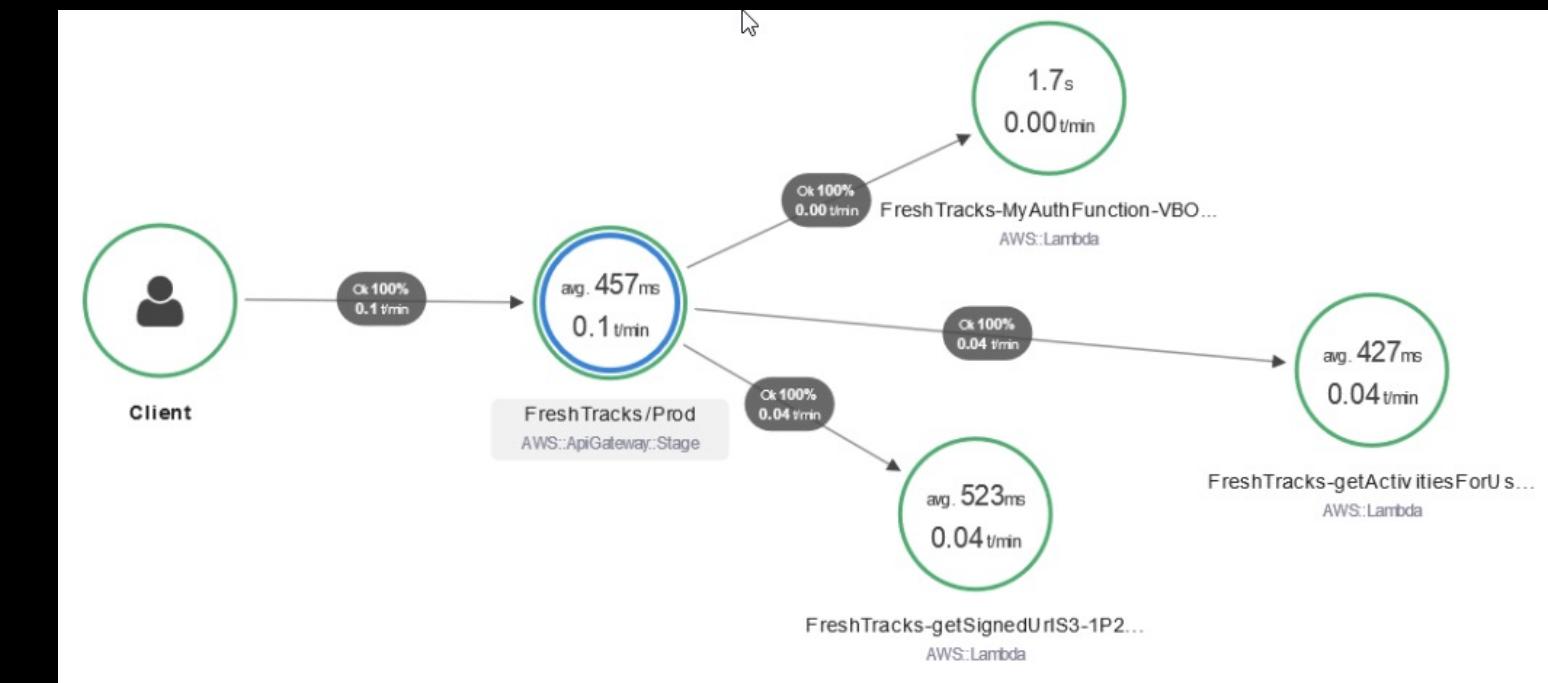
- API Gateway – inserts a tracing header into HTTP calls as well as reports data back to X-Ray itself



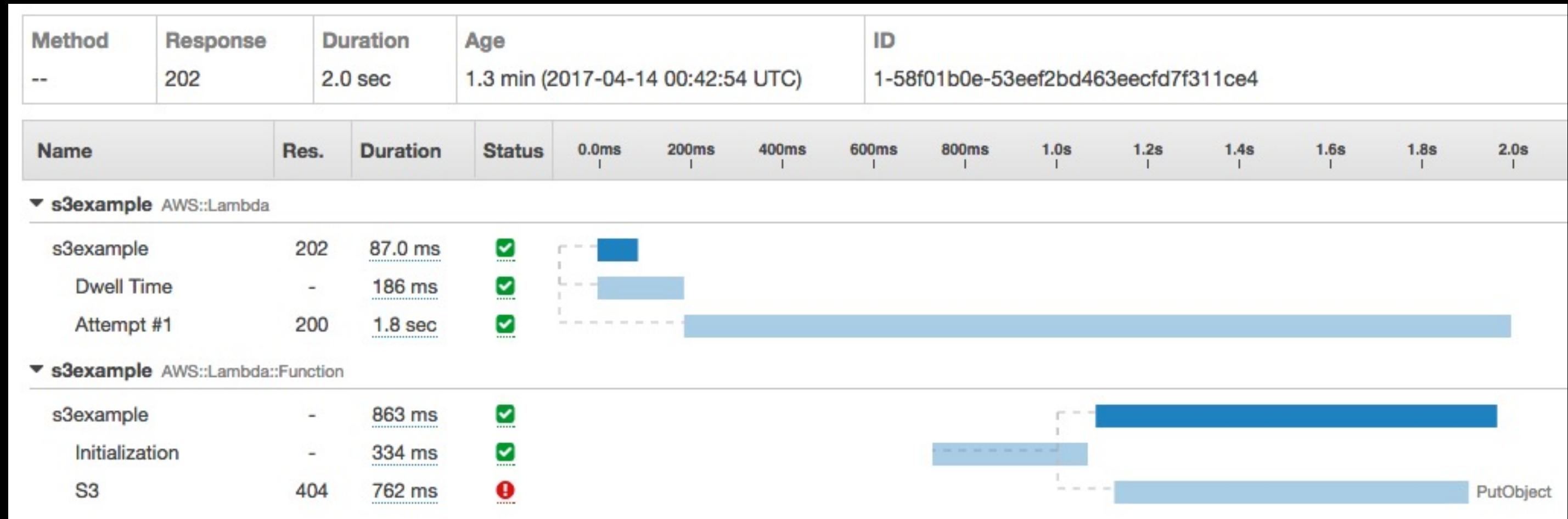
Globals:
Api:
TracingEnabled: True

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

const documentClient = new AWS.DynamoDB.DocumentClient();
```



X-Ray trace example



AWS X-Ray Analytics example



CloudWatch ServiceLens

CloudWatch ServiceLens

- Ties together CloudWatch metrics and logs, in addition to traces from AWS X-Ray
- Gives you a complete view of your applications and their dependencies

CloudWatch ServiceLens



Amazon CloudWatch

Metrics
Logs
Events
Alarms
Dashboards



AWS X-Ray

Traces
Analytics
Service map
Latency detection

Troubleshooting / query workflow

Notification / question

Receive a **CloudWatch alarm** notification

Ask a question

Identify ServiceLens

View **Service Lens** service map
Identify point of interest to dive deep

Traces X-Ray traces

View **X-Ray** traces, maps, and requests
Look at specific API / service that is the current point of interest

Analyze X-Ray Analytics

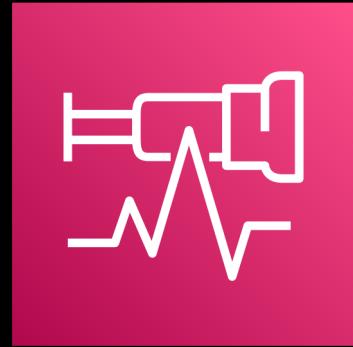
Trace analysis with **X-Ray Analytics**
Perform deep analysis / comparison of traces if necessary

Logs Logs Insights

Query **Logs Insights**
At specific point in time for deeper analysis and identify root cause

AWS open source observability services

AWS Open Source Observability Services



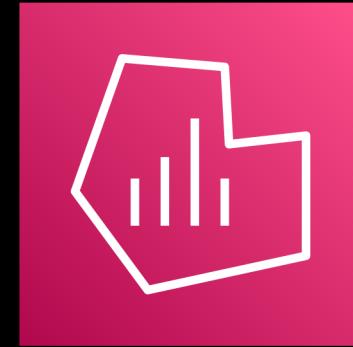
AWS Distro for
OpenTelemetry



Amazon Managed Service
for Prometheus



Amazon OpenSearch
Service



Amazon Managed Service
for Grafana

Collection

Metrics

Logs and Traces

Visualisation

What is OpenTelemetry?

- 50% of companies use at least 5 observability tools
- 33% of the companies used more than 10 observability tools
- Challenges
 - Using different SDK and agents
 - Increase in resource consumption
 - Manual correlation is error-prone
- OpenTelemetry is an opensource observability framework for cloud-native software. It is a collection of tools, APIs, and SDKs.
- You can use it to instrument, generate, collect, and export telemetry data (metrics, logs, and traces) for analysis in order to understand your software's performance and behaviour



AWS Distro for OpenTelemetry?

A SECURE, PRODUCTION-READY OPEN SOURCE DISTRIBUTION SUPPORTED BY AWS



AWS Distro for
OpenTelemetry

- Upstream-first distro of OpenTelemetry
- Certified by AWS for security and predictability
- Backed by AWS Support
- One-click deploy and configure from AWS container and AWS Lambda consoles
- The AWS Distro for OpenTelemetry collector is added as a layer to the lambda function
- Exporters for AWS monitoring solutions including – CloudWatch, X-Ray, Amazon Managed Service for Prometheus, OpenSearch Service and Partner Solutions

Resources

- Sample Serverless Application (Serverless Feedback App)
<https://github.com/aws-samples/aws-serverless-feedback-app>
- AWS Observability Workshop
<https://observability.workshop.aws>
- AWS Distro for OpenTelemetry
<https://aws-otel.github.io>
- Lambda Powertools - Python
<https://github.com/awslabs/aws-lambda-powertools-python>
- CloudWatch Embedded Metric Format
<https://s12d.com/cwl-emf-client>
- CloudWatch Metrics Explorer
<https://s12d.com/cw-me>
- Tracing AWS Lambda functions in AWS X-Ray with OpenTelemetry
<https://s12d.com/tracing-lambda-otel>
- Monitoring and observability – AWS Lambda Operator Guide
<https://s12d.com/lambda-op-guide-obs>
- Getting started with Trace Analytics in Amazon Elasticsearch Service
<https://s12d.com/trace-analytics-es>



Thank you!

Ozioma Uzoegwu
Solutions Architect, AWS

