# Serverless Observability

Heitor Lessa
Specialist Solutions Architect
AWS

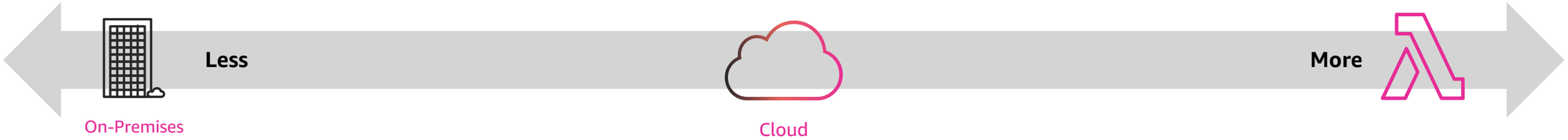
@heitor_lessa

Ran Ribenzaft
CTO
Epsagon

@ranrib

aws Webinar

# AWS operational responsibility models

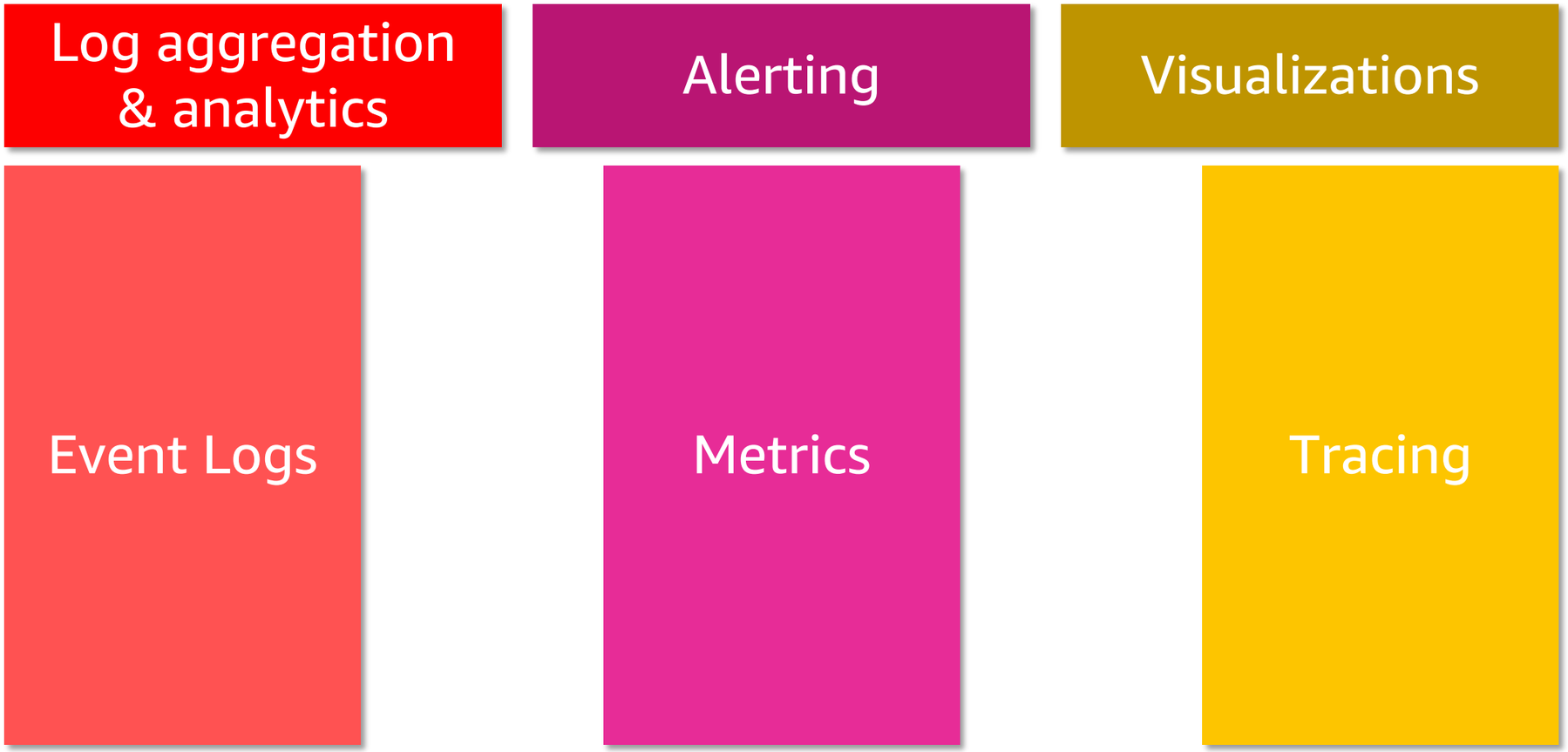



| | | | | | | |
|---|---|---|---|---|---|---|
| **Compute** | Virtual Machine | Amazon EC2 | AWS Elastic Beanstalk | | Fargate | AWS Lambda |
| **Databases** | MySQL | MySQL on EC2 | Amazon RDS MySQL | Amazon Aurora | Amazon Aurora Serverless | Amazon DynamoDB |
| **Storage** | Storage | | | | | Amazon S3 |
| **Messaging** | ESBs | | Amazon MQ | | Amazon Kinesis | Amazon SNS / SQS |
| **Analytics** | Hadoop | Hadoop on EC2 | Amazon EMR | | Amazon Elasticsearch Service | Amazon Athena |

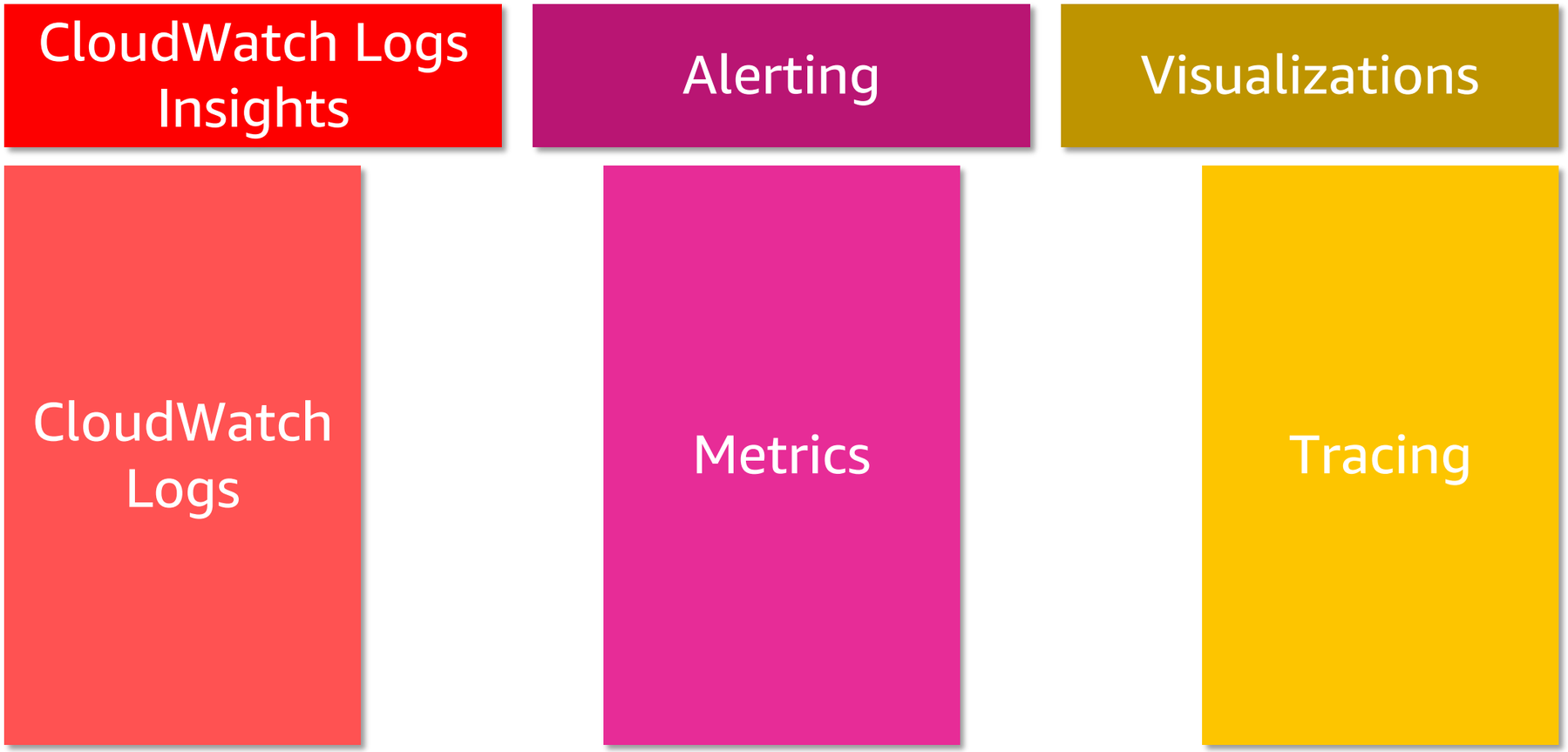
# The Three Pillars of Observability


**Distributed Systems Observability** *by Cindy Sridharan*

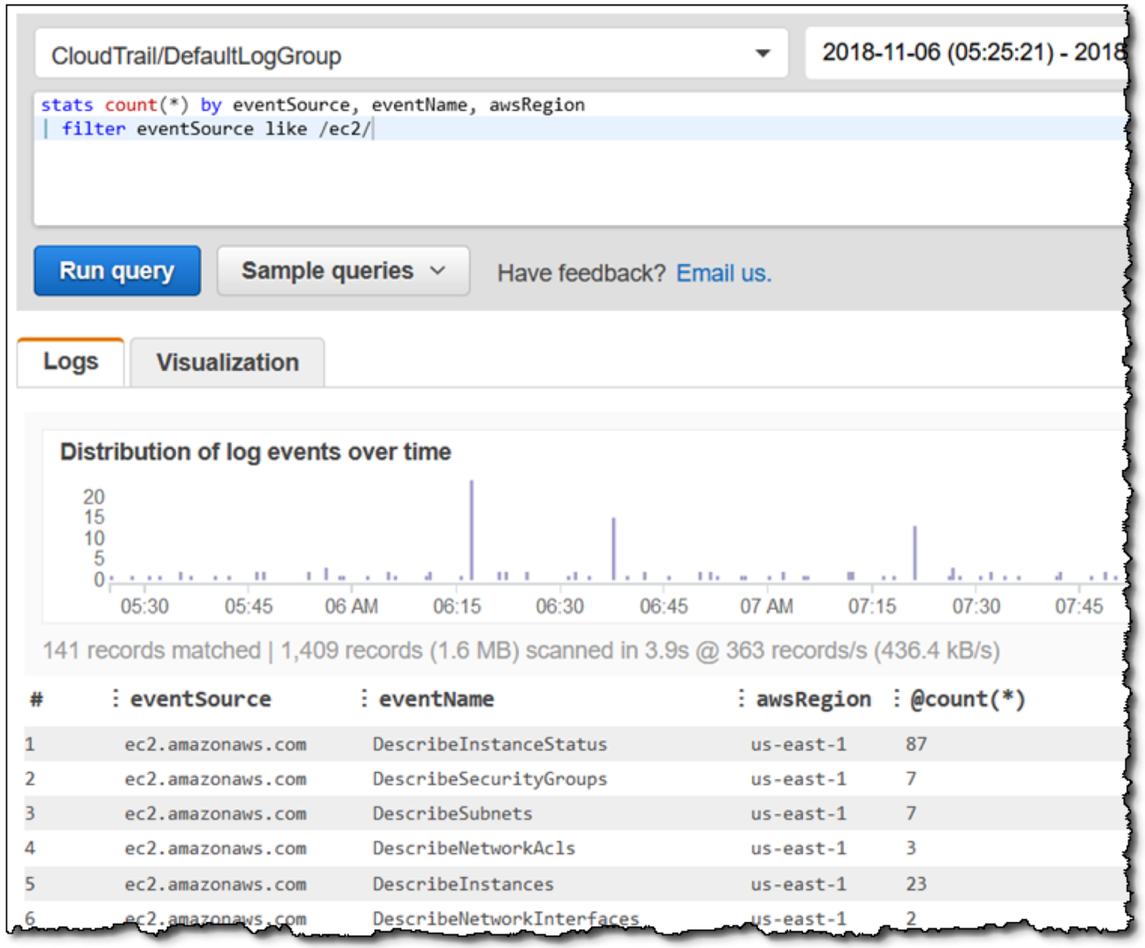
# Using Observability

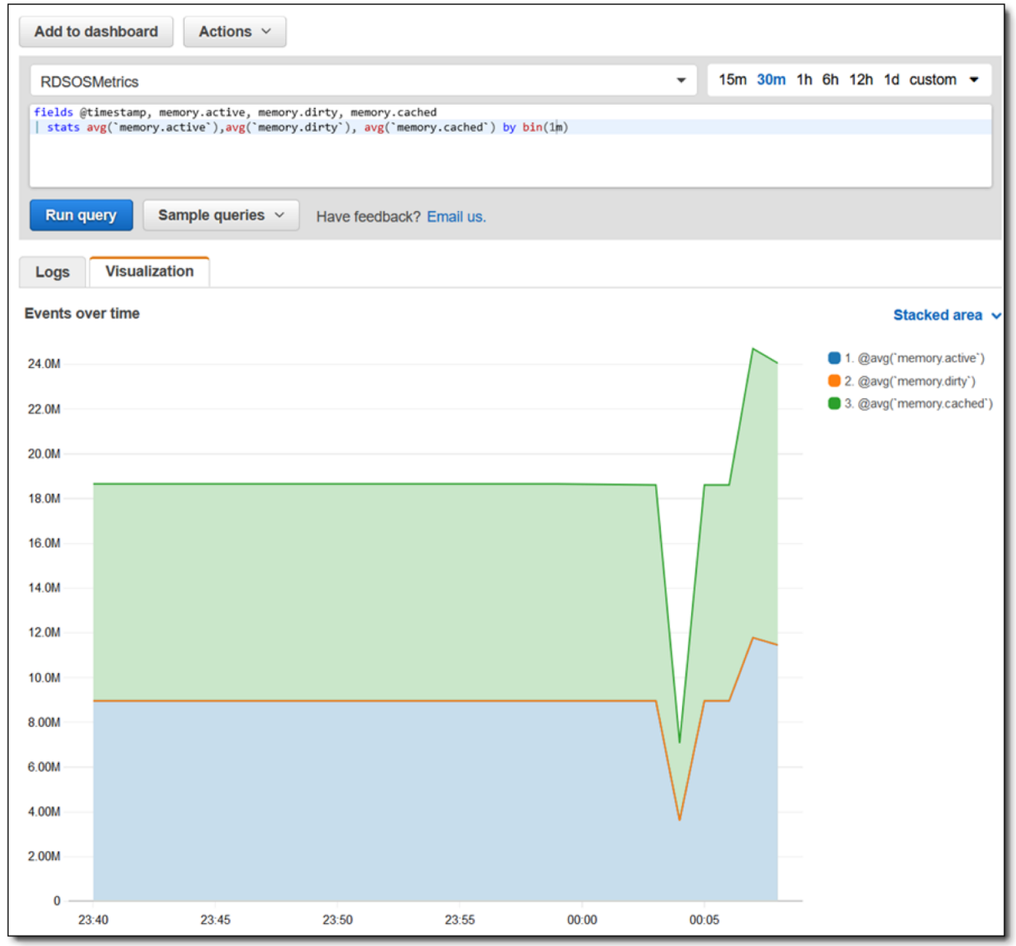


# Using Observability

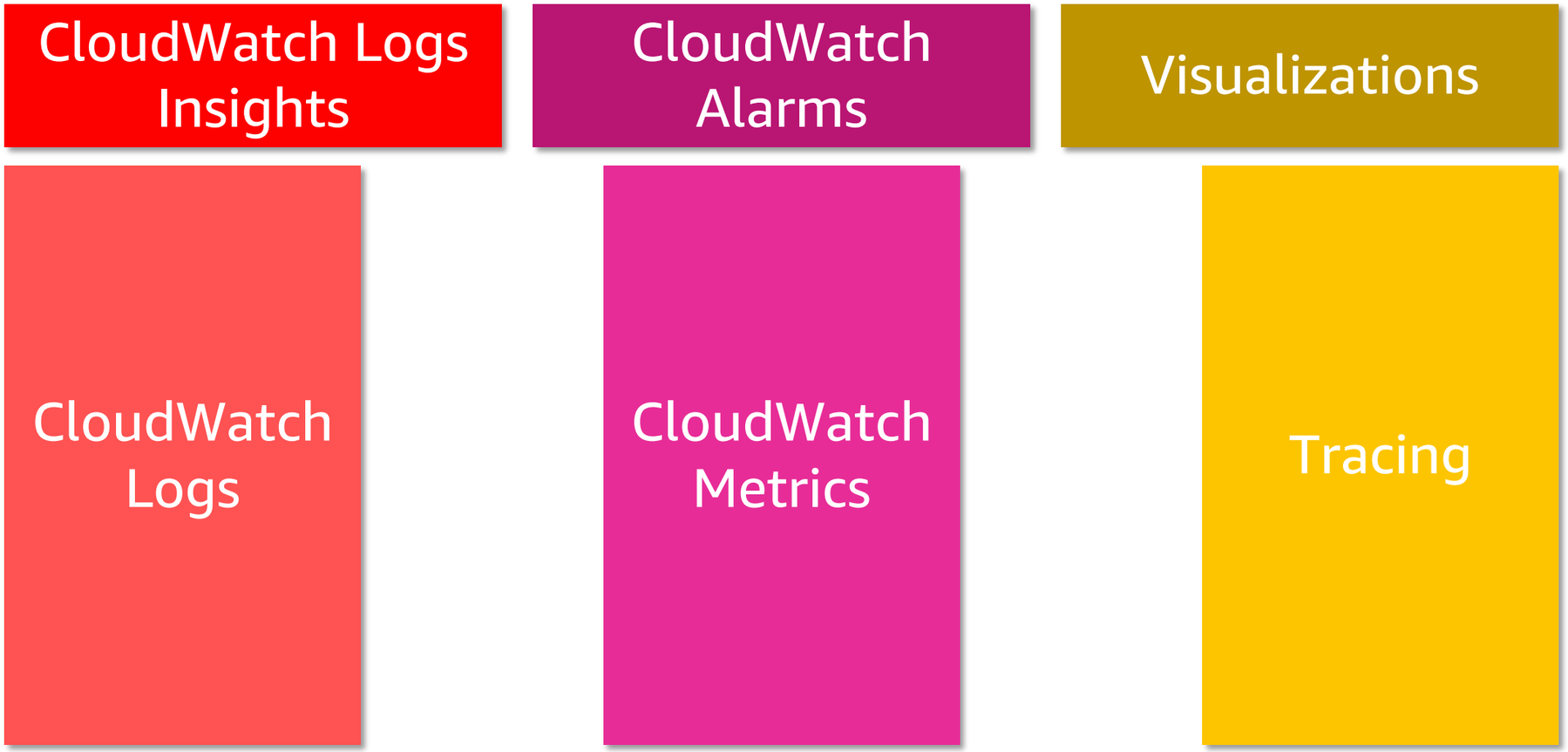


# Using Observability – CloudWatch Logs, Insights



# Using Observability

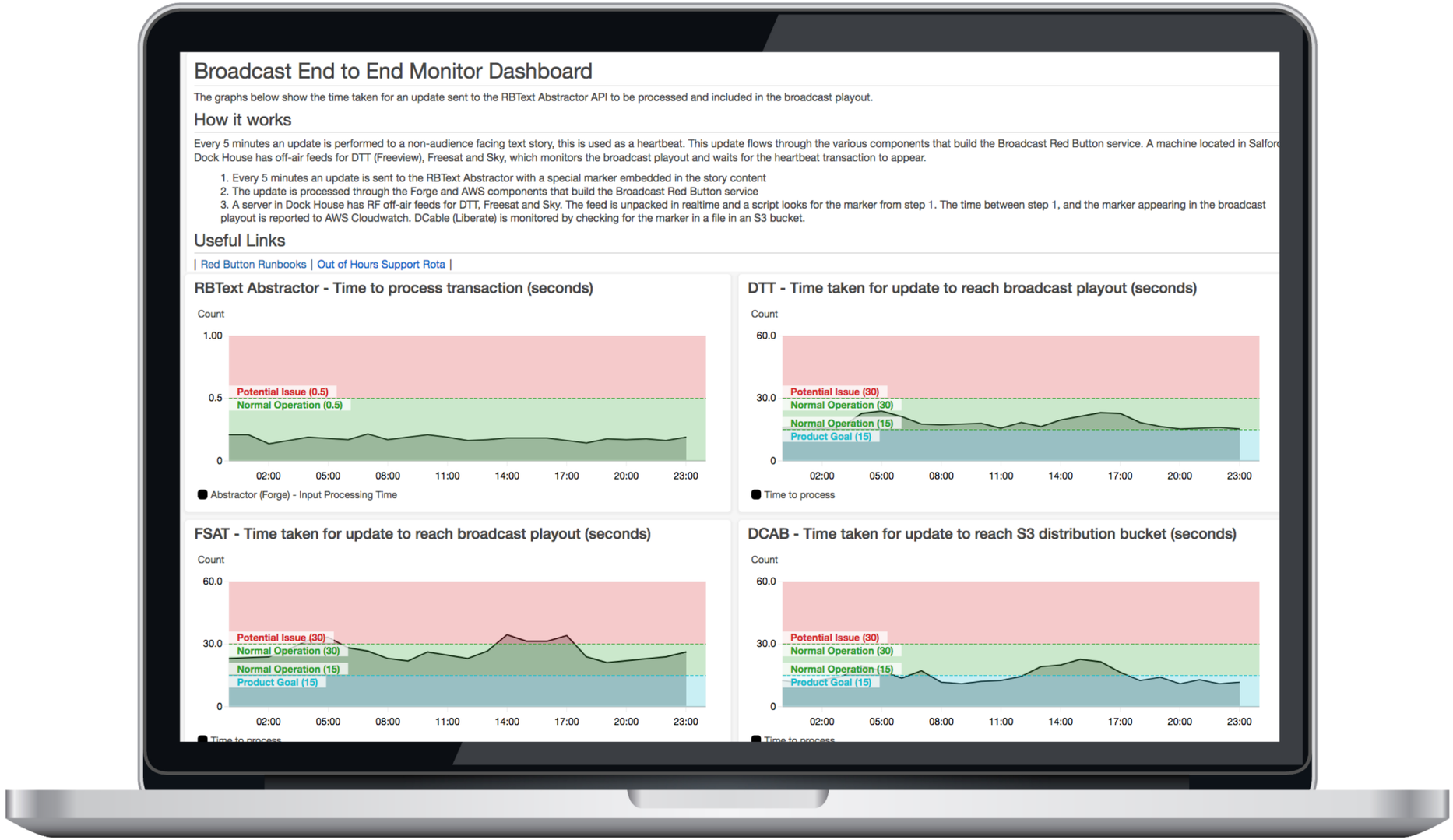


# Using Observability – CloudWatch Custom Metrics

```
const metricData = await cloudWatch.putMetricData({
        MetricData: [
            {
                MetricName: 'My Business Metric',
                Dimensions: [
                    {
                        Name: 'Location',
                        Value: 'Paris'
                    }
                ],
                Timestamp: new Date(),
                Value: 123.4
            }
        ],
        Namespace: METRIC_NAMESPACE
    }).promise();
```

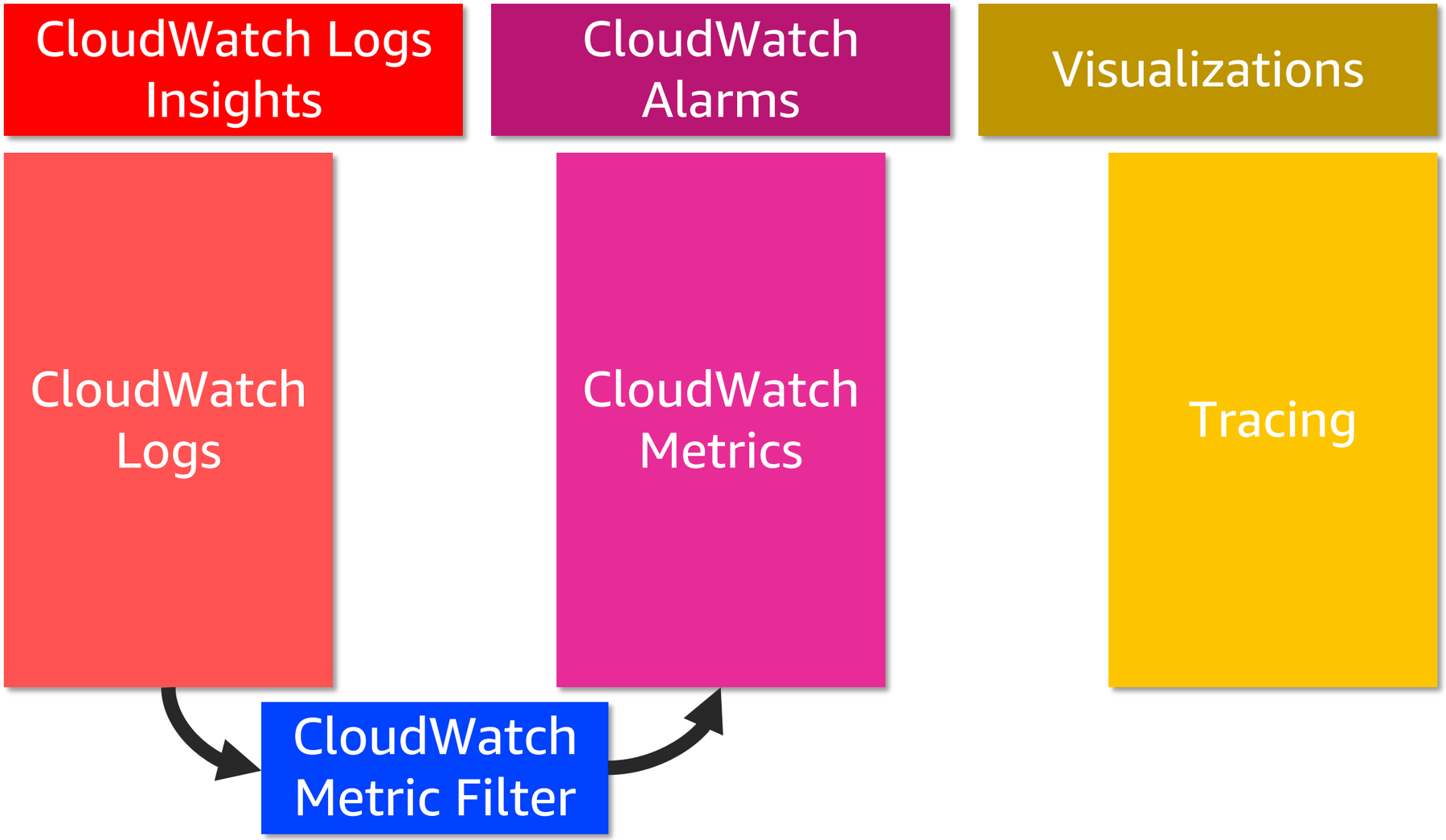
- Metric name
- Dimensions
- Timestamp
- Value
- Namespace

# Using Observability – CloudWatch Metrics/Dashboard

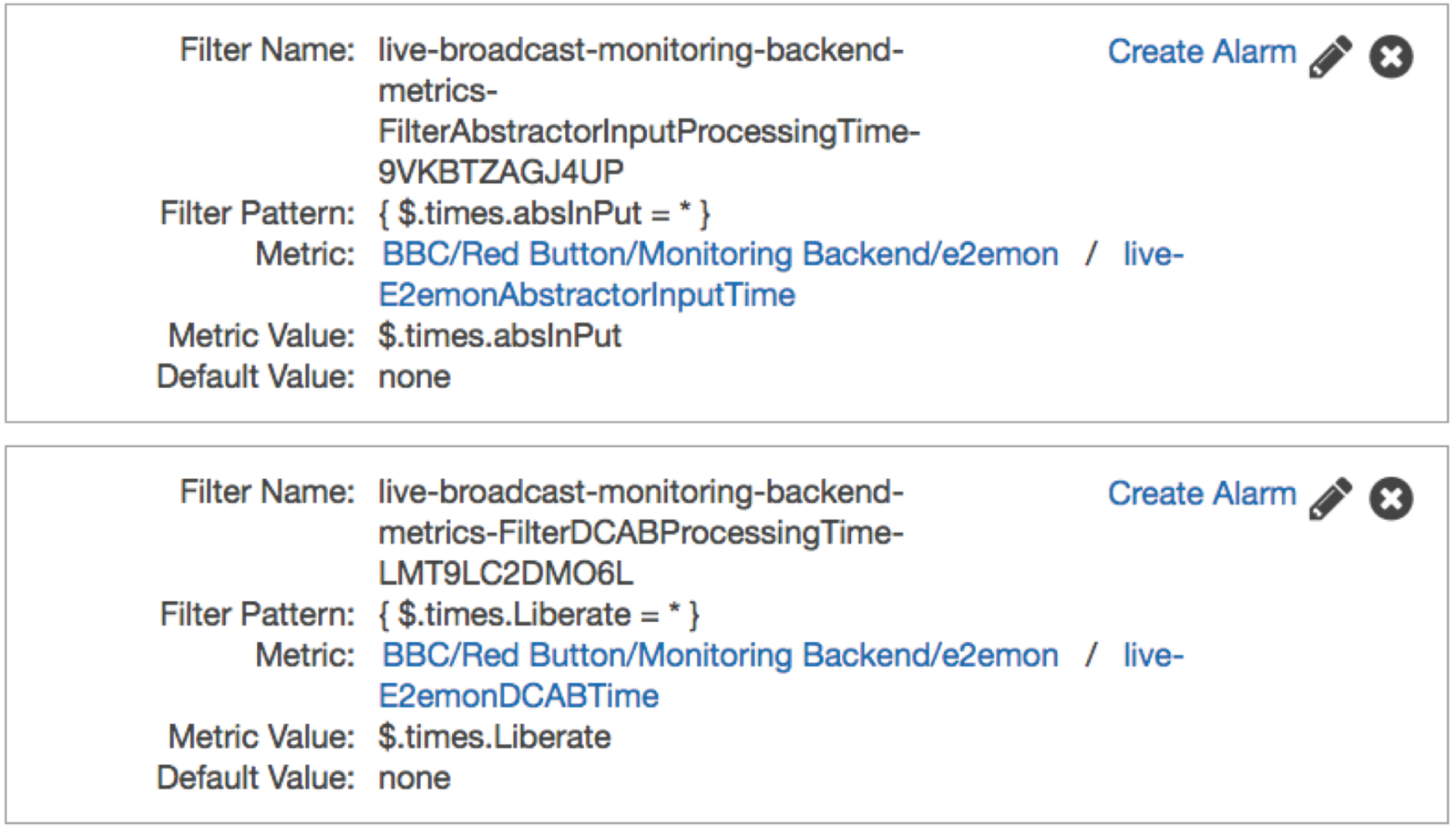



**[DEV 302] Monitor All Your Things**: Amazon CloudWatch in Action with BBC

# Using Observability

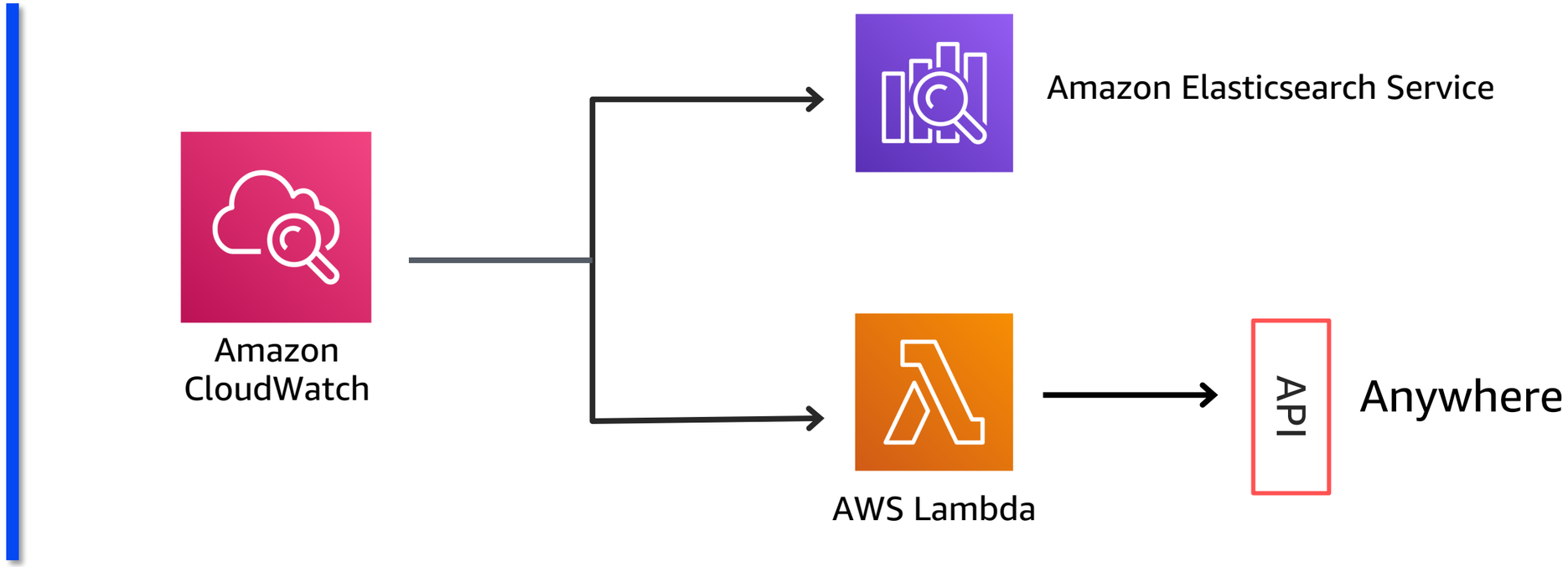


# Using Observability – CloudWatch Logs Metric Filter

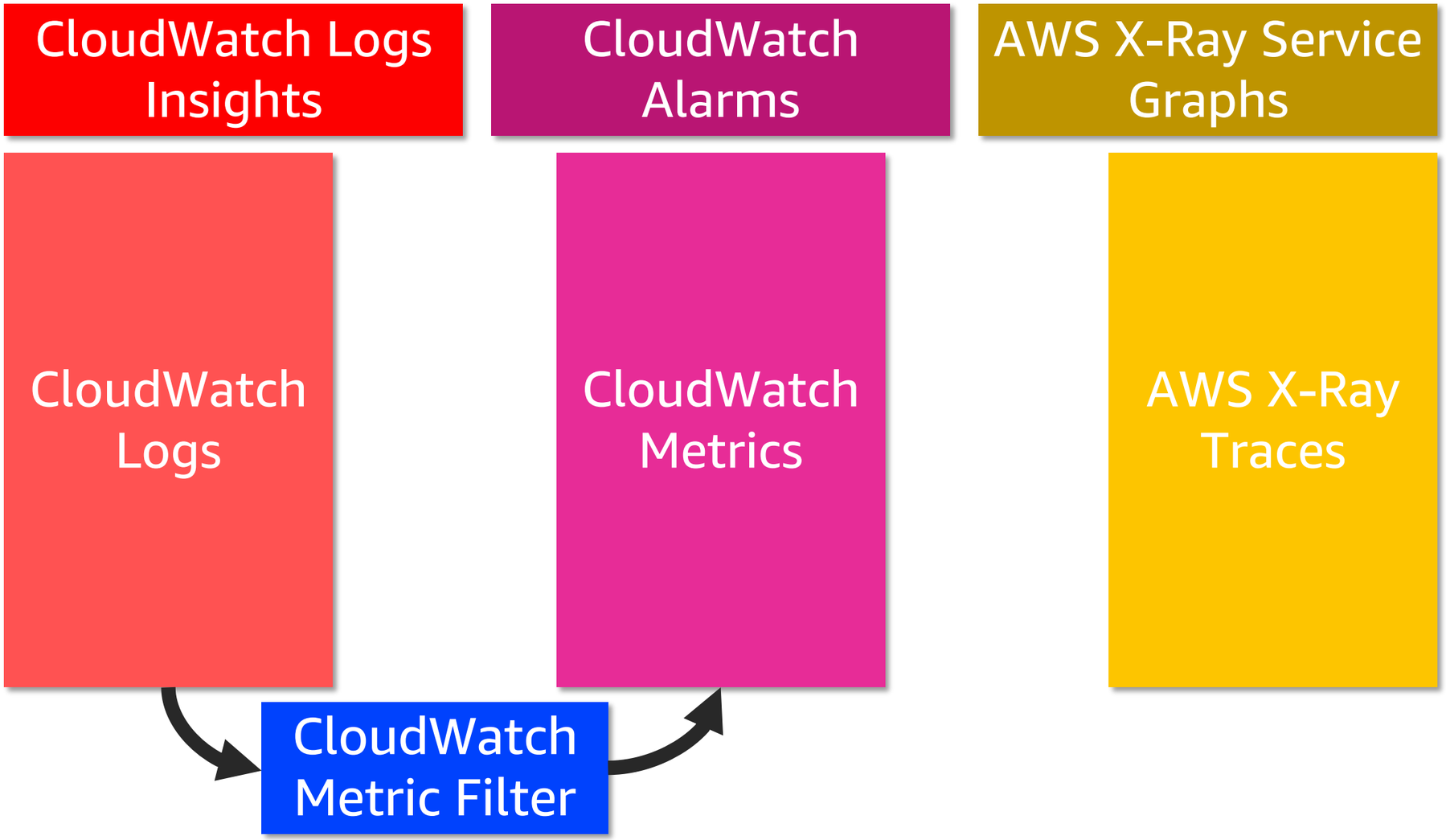



**[DEV 302] Monitor All Your Things**: Amazon CloudWatch in Action with BBC

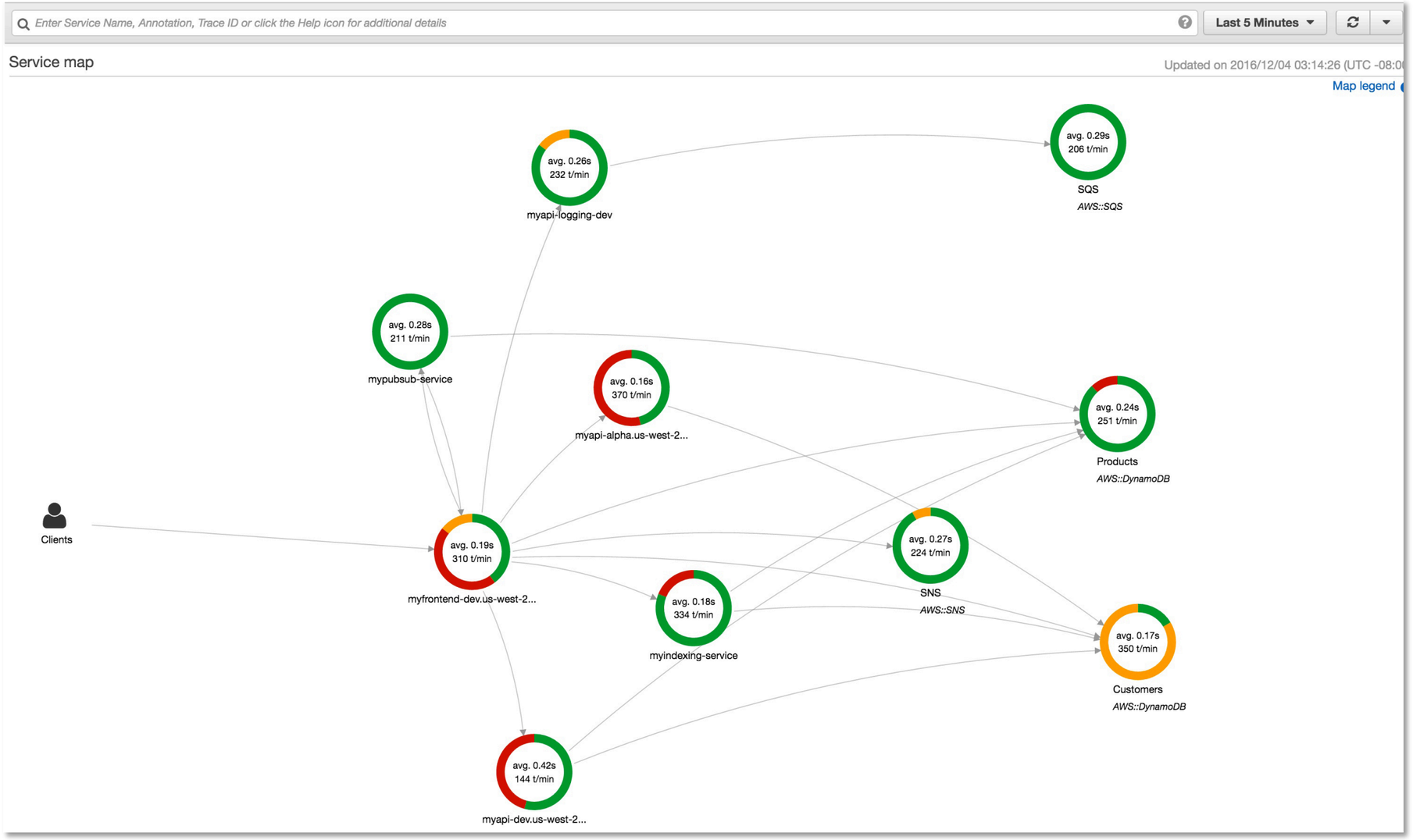
# Using Observability – CloudWatch Logs Subscription



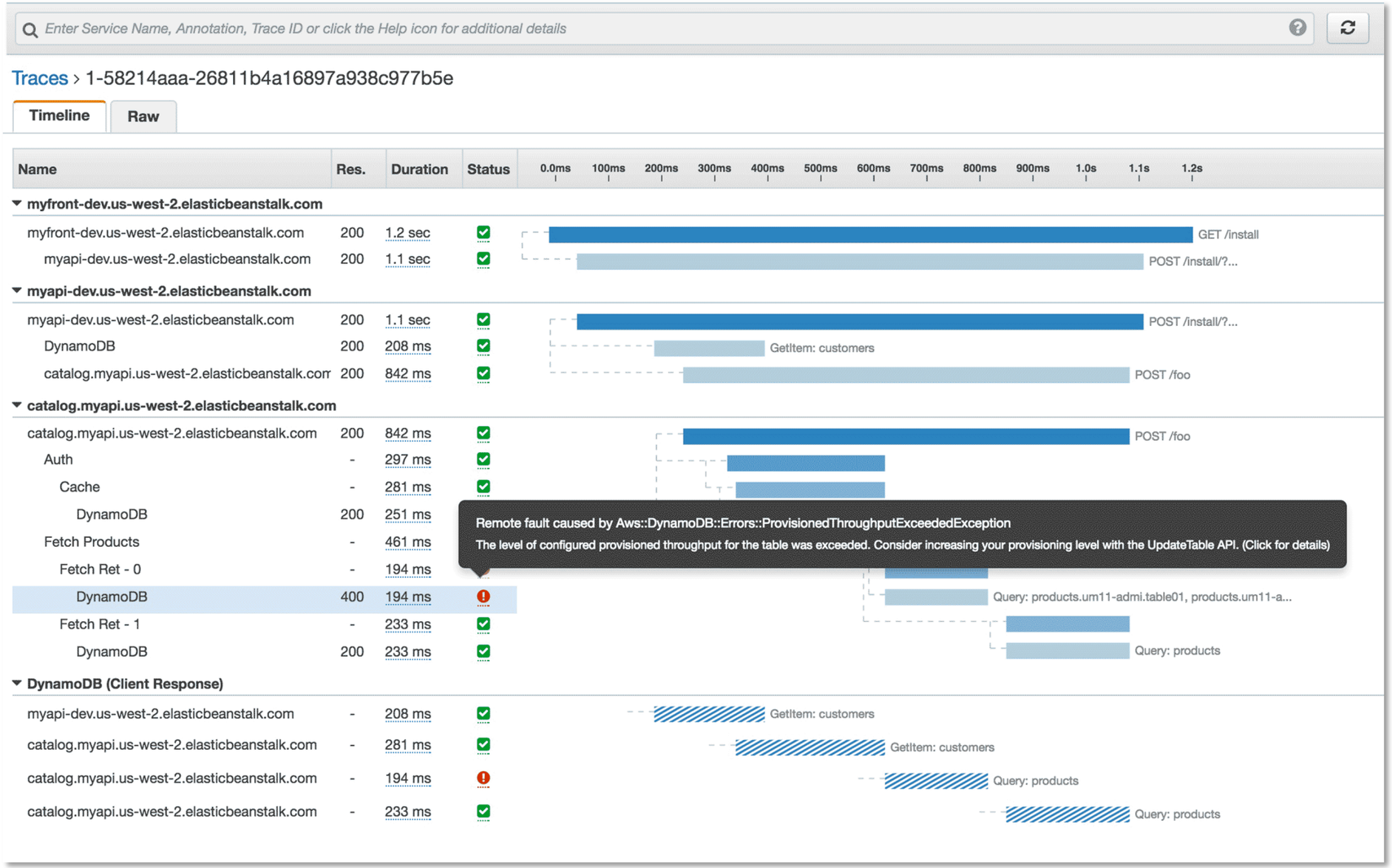
# Using Observability



# Using Observability – AWS X-Ray Service Graph



# Using Observability – AWS X-Ray Traces




***The story of the request***


THE FUTURE OF CHAOS?

# The Serverless Shop

aws Webinar

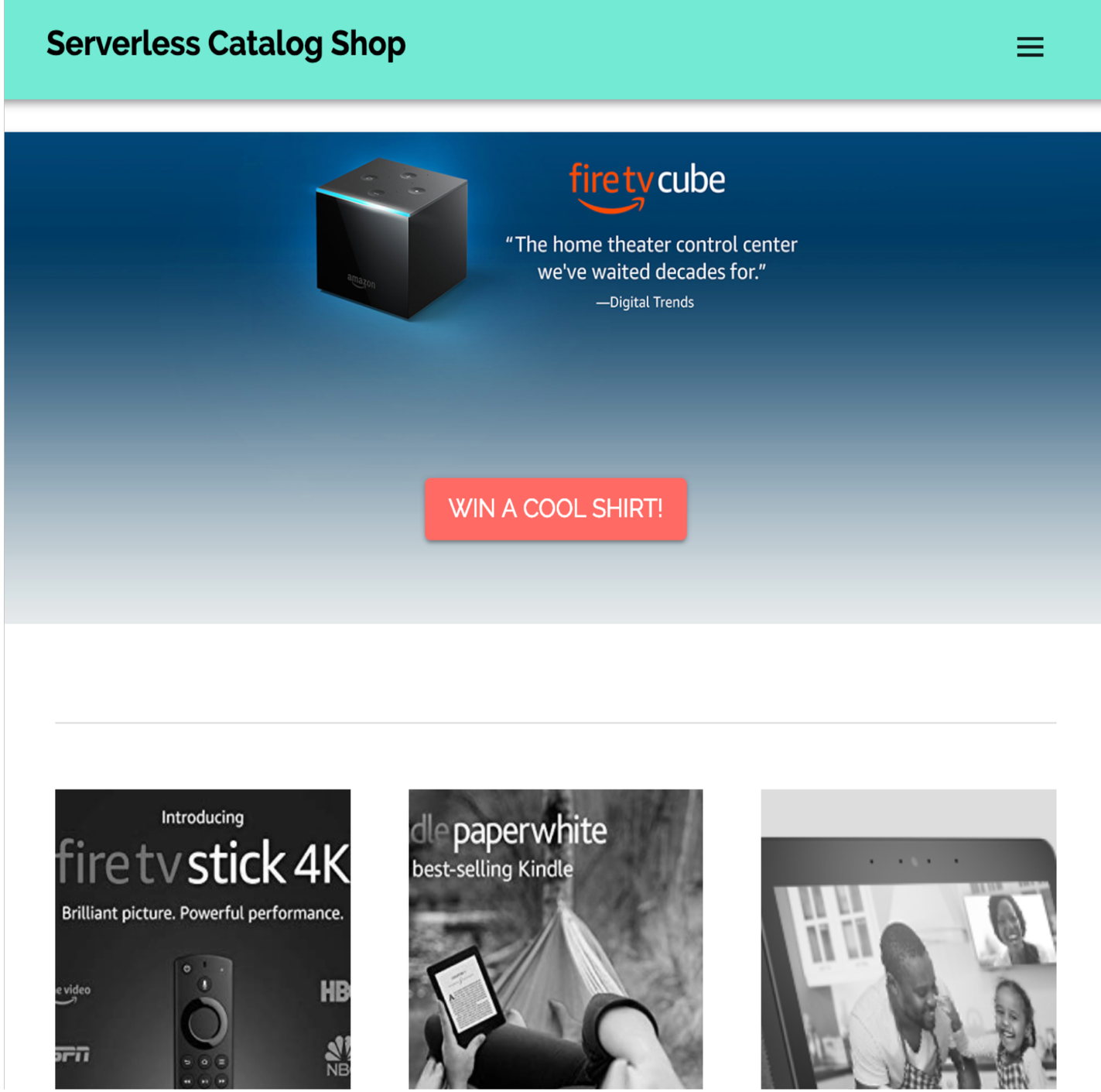
# The Serverless Shop

Simple e-commerce example

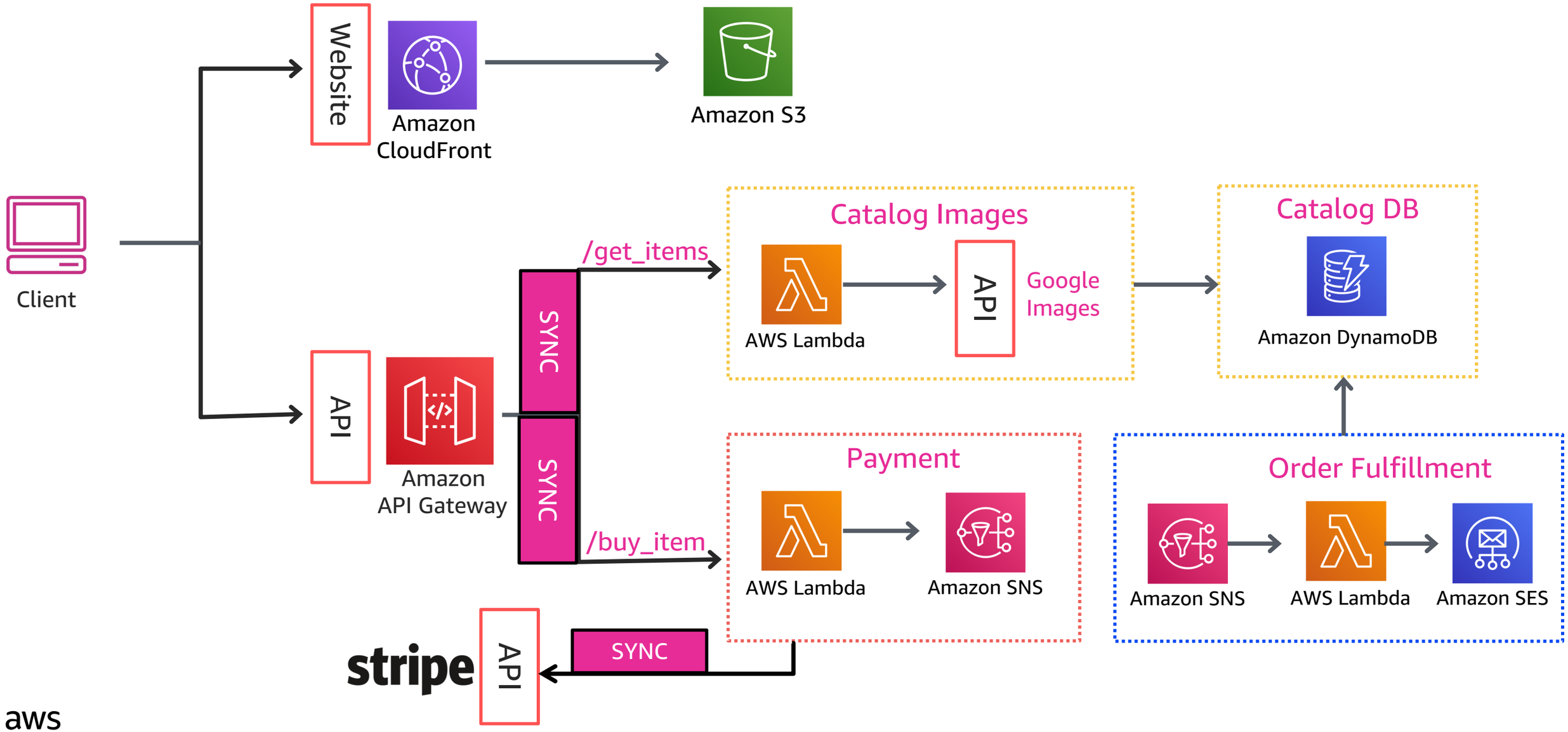
Browse and purchase products

Built 100% on Serverless resources

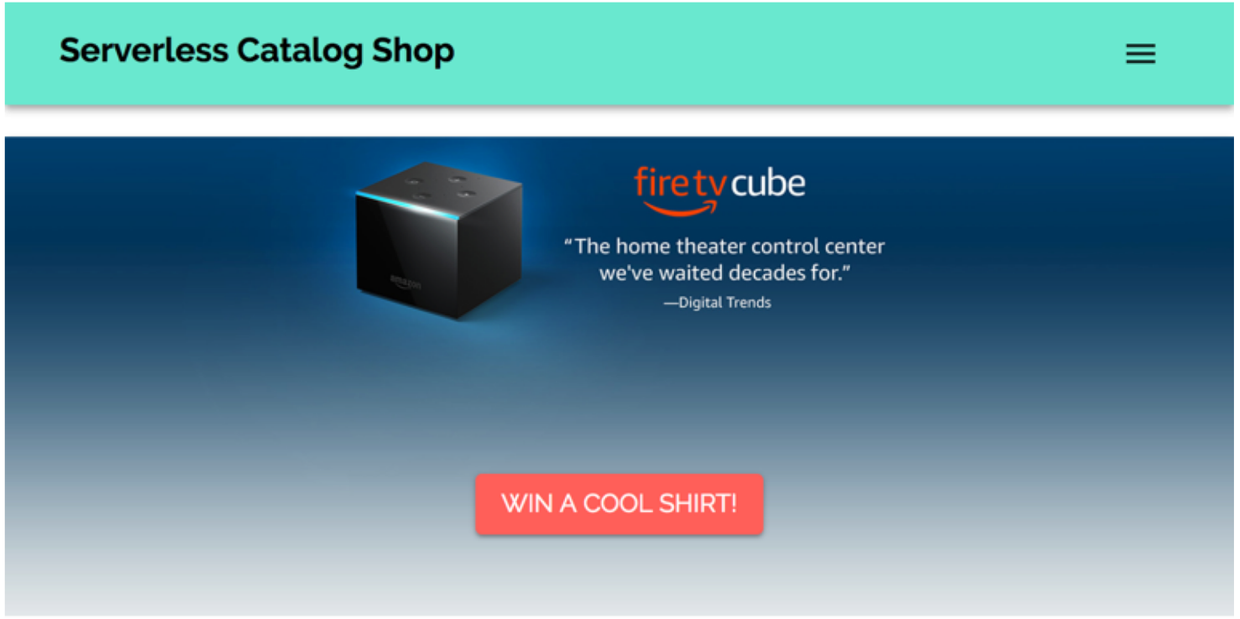
https://store.epsagon.io



# The Serverless Shop: Architecture



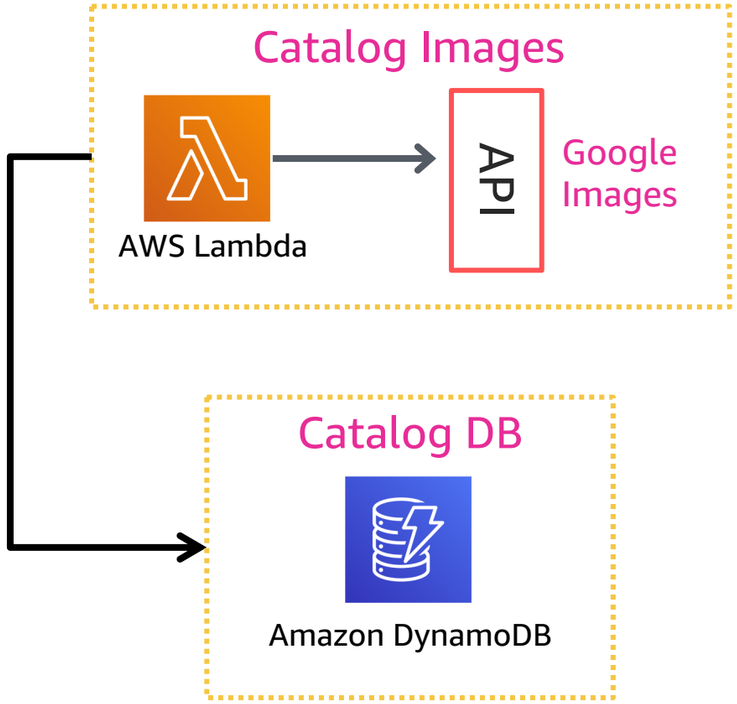
# The Serverless Shop: Catalog feature



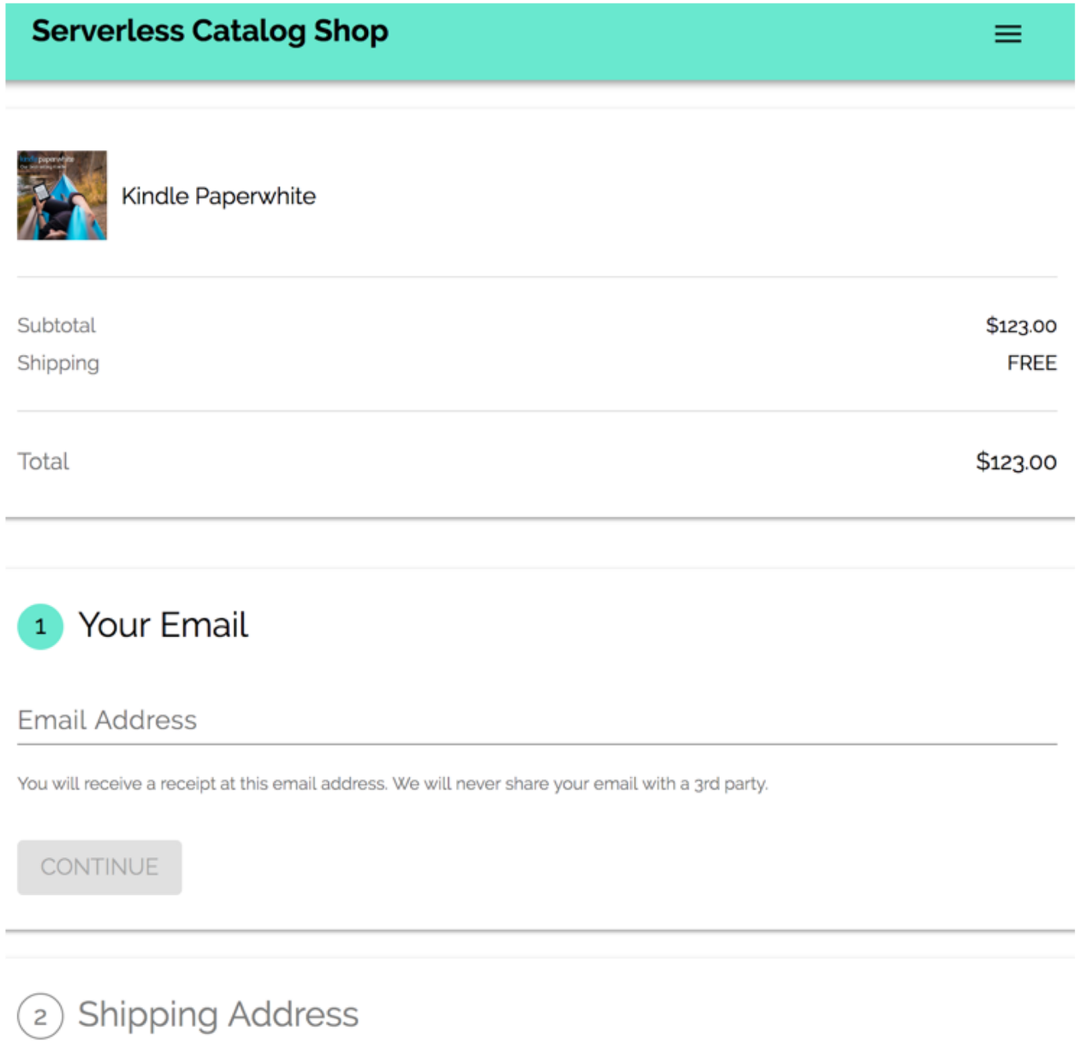

Fire TV Stick 4K
$123

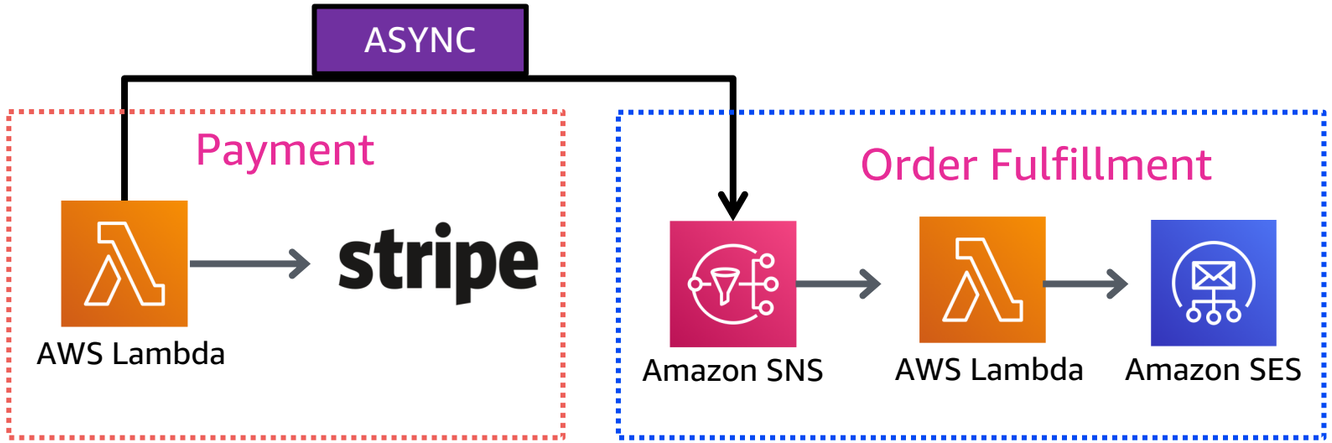

Kindle Paperwhite
$123

Echo Dot (3rd Gen)
$50



# The Serverless Shop: Payment feature



# Business flows and KPIs

## Catalog

### Time to Catalog

How long it takes for customers to see our products

## Payment

### Time to purchase

How long it takes from checkout to confirmation mail

# Let’s go shopping!

aws Webinar

# Observability in action: Scenarios

## Growth

**Catalog grows by N**

Image searching becomes a bottleneck

## No stock

**Runs out of products**

And yet purchase is successful

# Summary

# Challenges of Serverless applications

- No access to the underlying infrastructure that runs the code
- Troubleshooting event driven applications is hard
- Cost calculation requires understanding of business transactions
- Ephemeral and limited compute duration

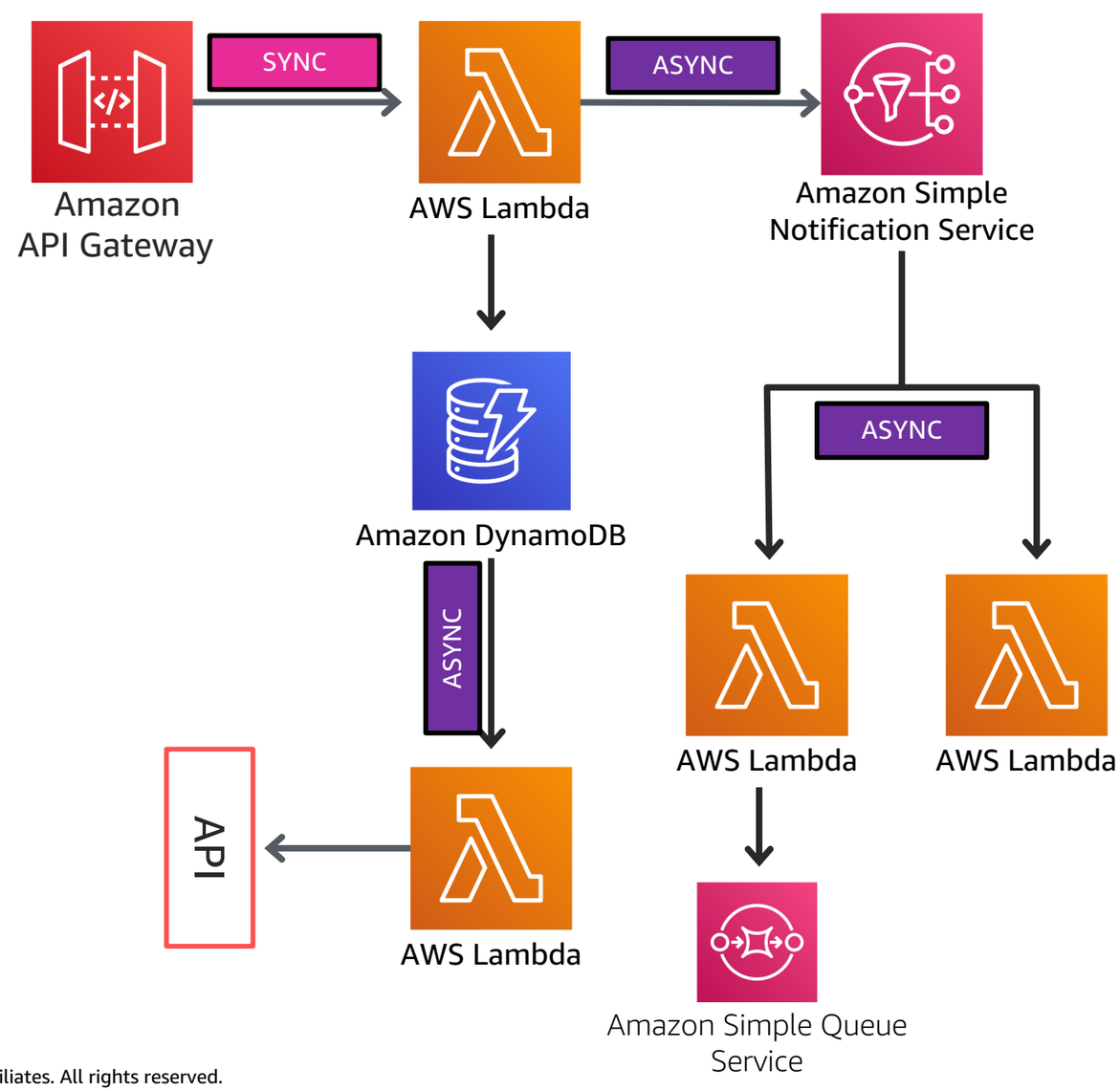
# Challenges of ~~Serverless~~ distributed applications

Managing lots of resources and APIs

End to end performance

Correlation between logs

Distributed errors



# Q&A

aws Webinar

# Appendix

# Some ideas you can use to improve the shop

- Lazy loading images (front-end)
- Check stock before allowing purchase
- Structured logging
- Custom metrics (CloudWatch API) for KPIs
- Order service
- Metrics from front-end
- DLQs

# Scenario #1: store grows

- More and more items are being added to the store
- Searching for images becomes a bottleneck as it’s done on the back-end
- Result -> timeout -> user can’t see any item in the store

# Scenario #2: stock run out

- 2 customers try buying the last piece of an item
- Both get the confirmation screen, only one get’s a success mail
- Result -> Un-happy customer
- Mis-handled request leads to business impact

# Example Epsagon tracing - Payment