Comprehensive Observability via Distributed Tracing

Chinmay Gaikwad, Technical Evangelist

Twitter: @epsagon

epsagon

Hello!

- Software Engineer, Applications Engineer, Technical Marketing Engineer: Intel, IBM, early stage startups

- Traveling, Soccer, Restaurants, Video Games

What we’ll discuss today

- Microservices: The New Normal and New Challenges
- Troubleshooting Distributed Environments
- Benefits of Distributed Tracing

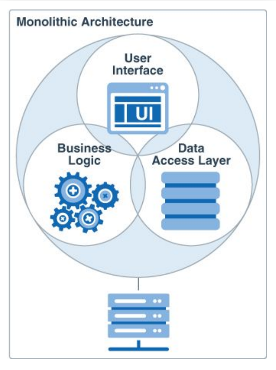
WHY?

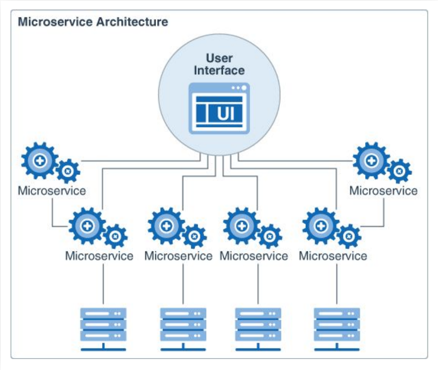
HOW?

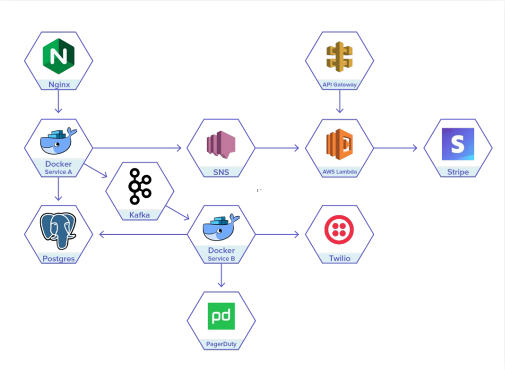
Microservices: The New Normal and New Challenges

epsagon

The Rise of Microservices





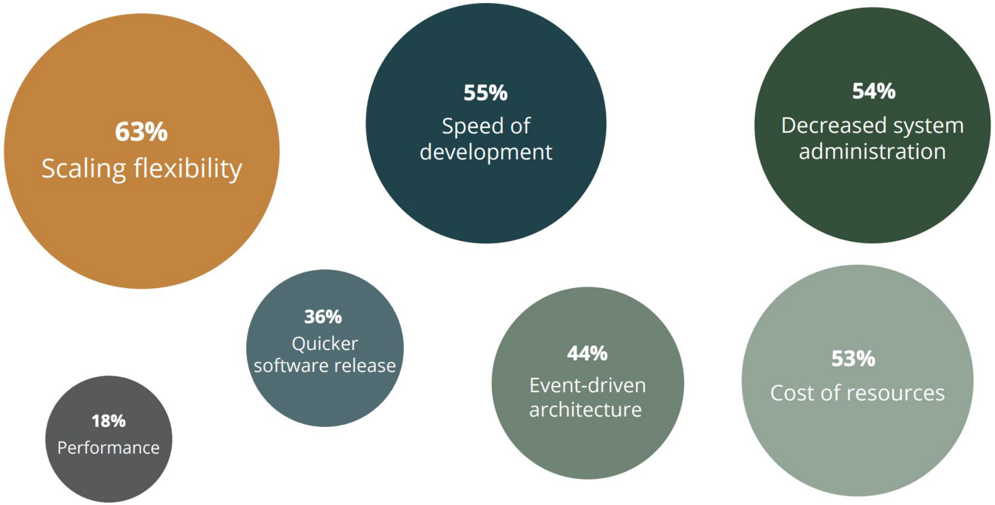


Host-based Monolithic → Host-based Distributed → Abstracted-host Highly Distributed

Extremely hard to monitor and troubleshoot

Why Microservices?

What are the biggest benefits of using Serverless for your organization?



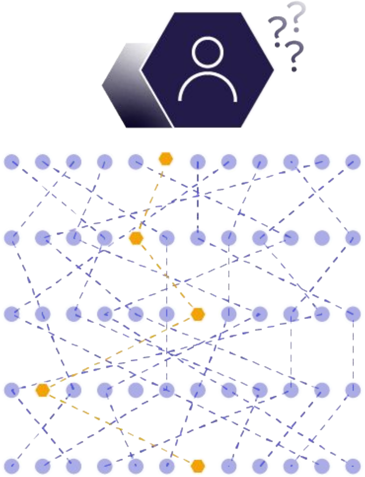
Source: State of Serverless Report 2020 - CodingSans

New Paradigm, New Challenges

Difficulty Identifying & Troubleshooting Issues
Customer-facing impacts (downtime, latency)
Decreased velocity of new feature releases

Traditional Monitoring from Multiple Sources
Lack of application insights & visibility into errors
Difficulty correlating data

Incomplete Data Insights
Sampling, resulting in gaps
No visibility into payloads



Troubleshooting Distributed Applications

epsagon

Observability: Overview

- Observability: Actively debug a system
- Monitoring: Watch and understand the state of a system
- Monitoring and observability is one of a set of capabilities that drive higher software delivery and organizational performance
- Who is monitoring and observability for? Everyone!

Source: DORA research

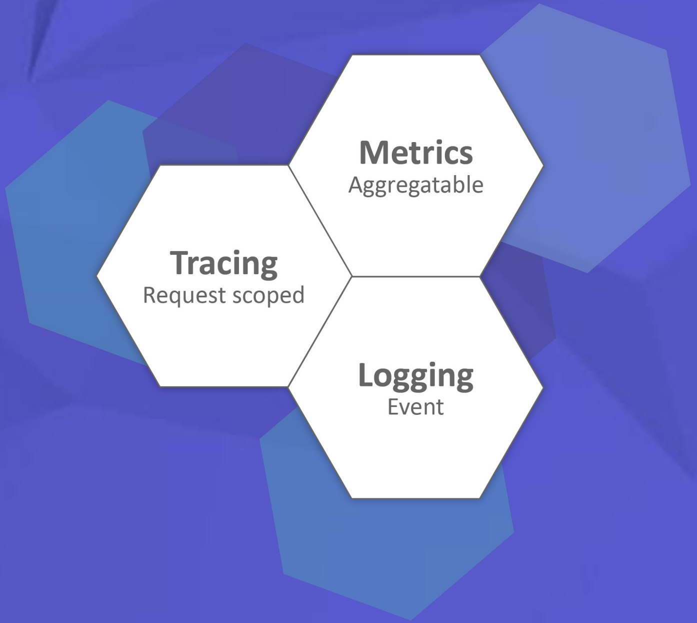
Achieving Observability in Microservices

Combining metrics, logs, and traces for observability is the only way to understand complex environments

Metrics tell us the “**what**”

Logs tell us the “**why**”

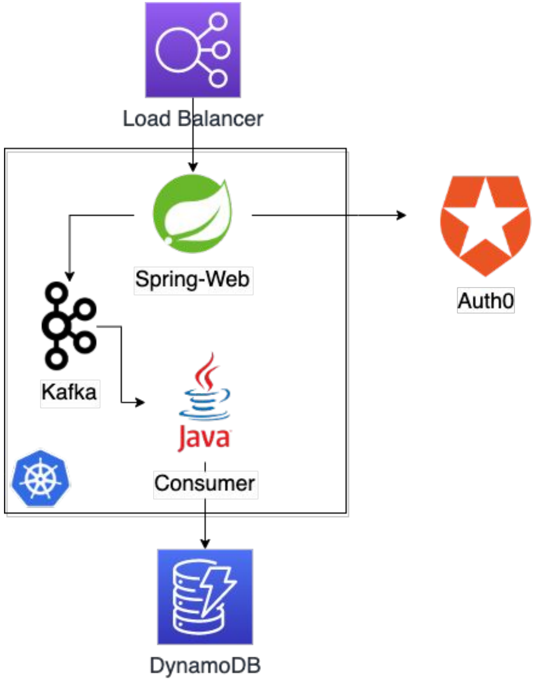
Traces tell us the “**where**”



epsagon

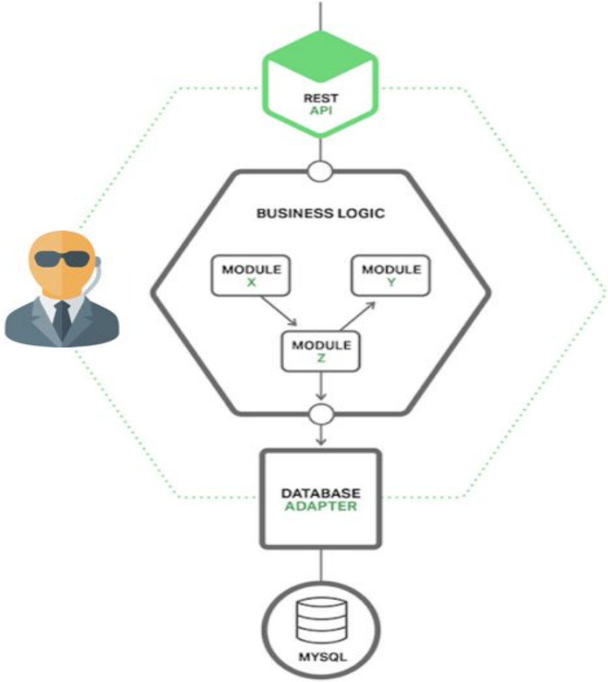
Distributed Troubleshooting Use Case

- The service implements a simple **virtual shop**, where users can send orders for items
- The **HTTP server** authenticates requests using **Auth0 API** (3rd party) and pushes them to a **Kafka** stream
- Another **Java container** polls the stream and updates the orders on a **DynamoDB** table
- Both containers run + Kafka stream runs on **Kubernetes**
- Users complain about orders that were sent but not handled

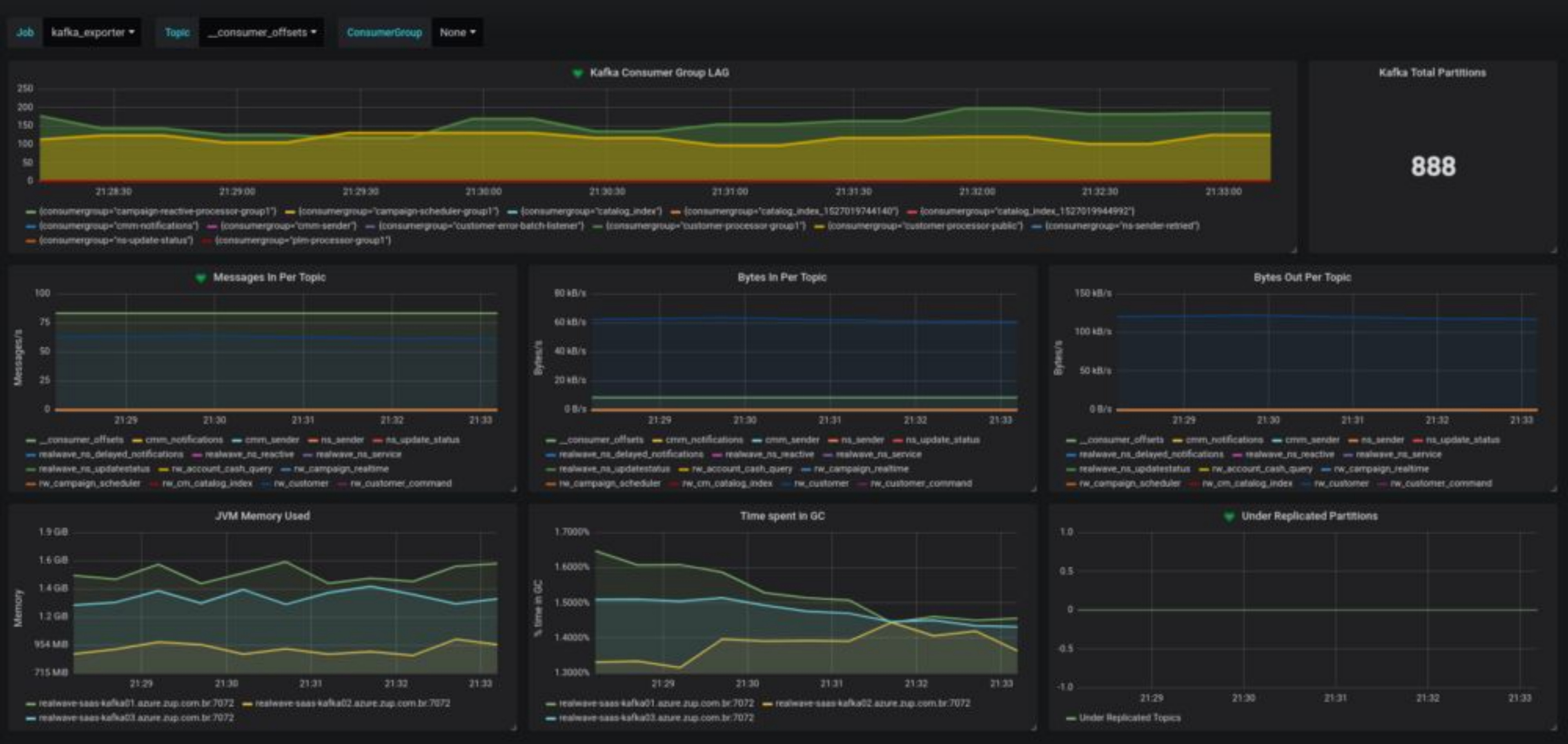


Old School Monitoring

- Heavy Instrumentation
- Collects only host data
- Collects only metrics

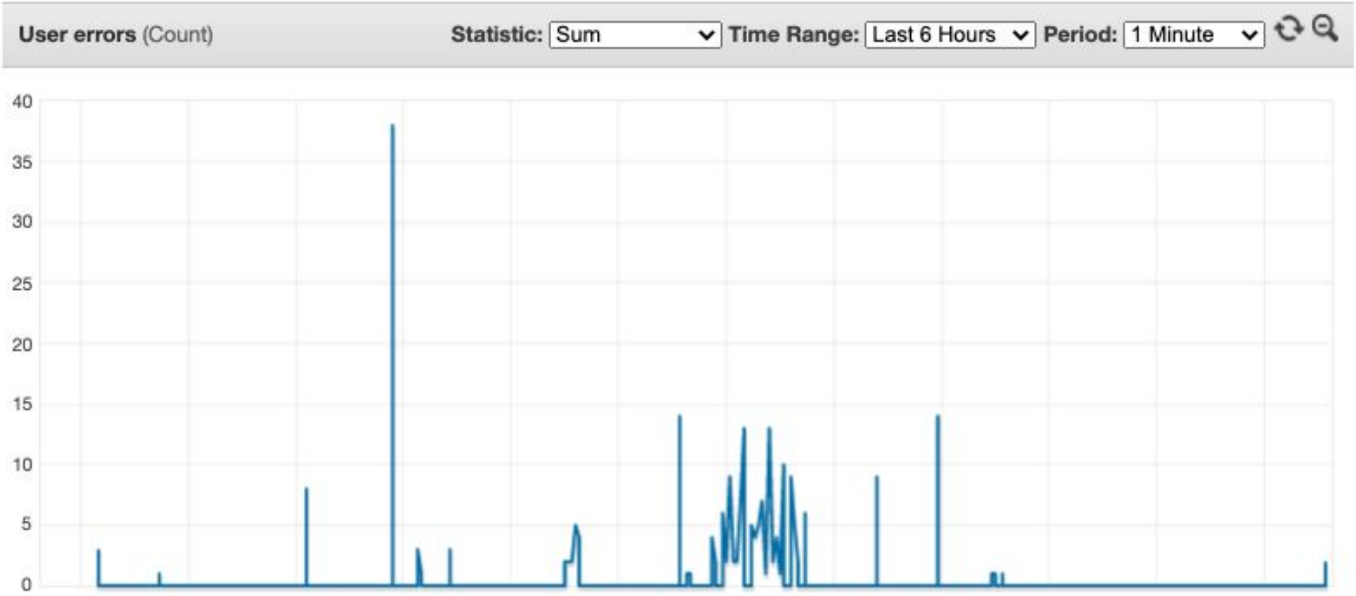


Kafka Metrics



Job kafka_exporter ▾ Topic __consumer_offsets ▾ ConsumerGroup None ▾

DynamoDB Metrics



User errors (Count)

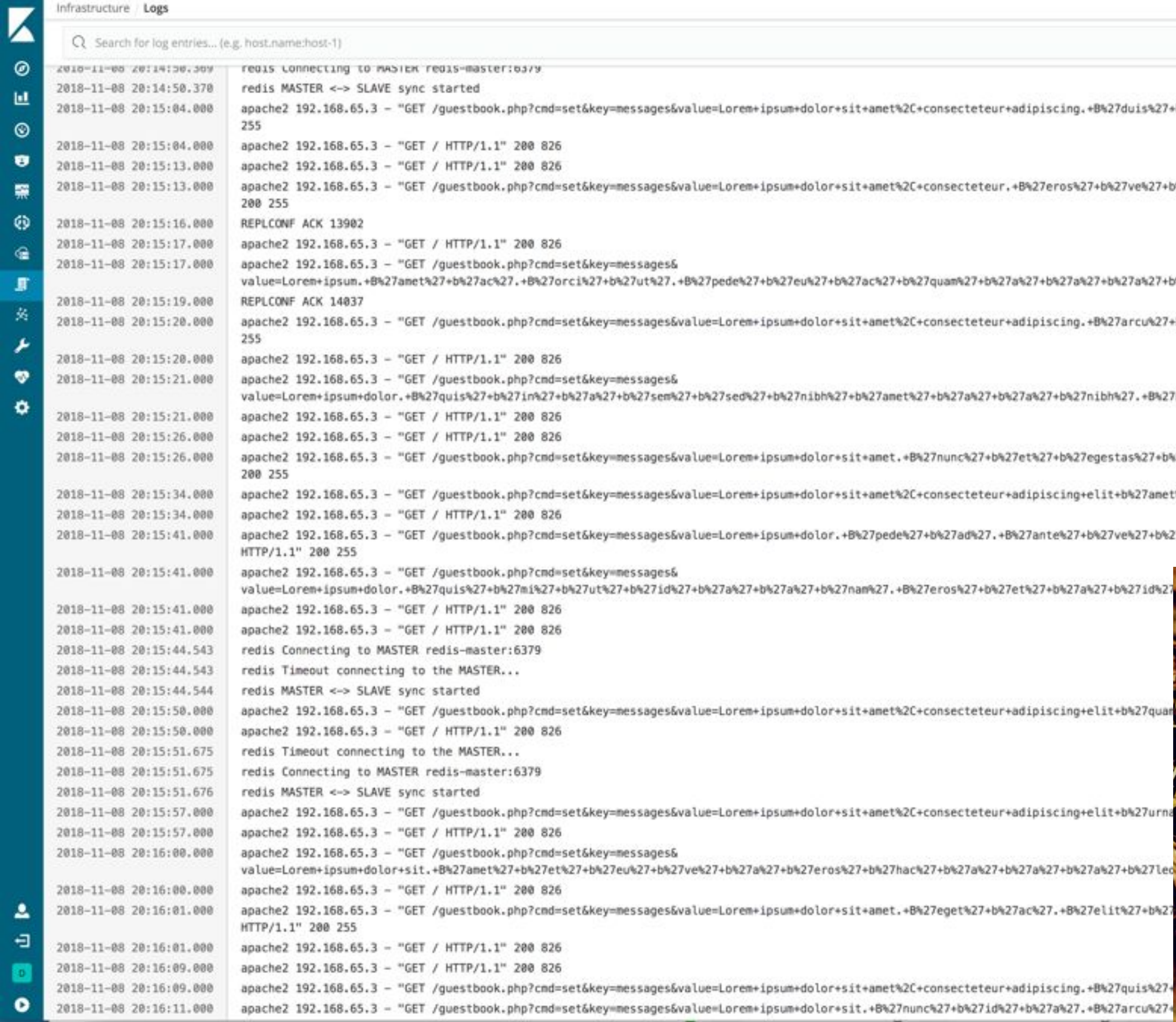
Statistic: Sum

Time Range: Last 6 Hours

Period: 1 Minute

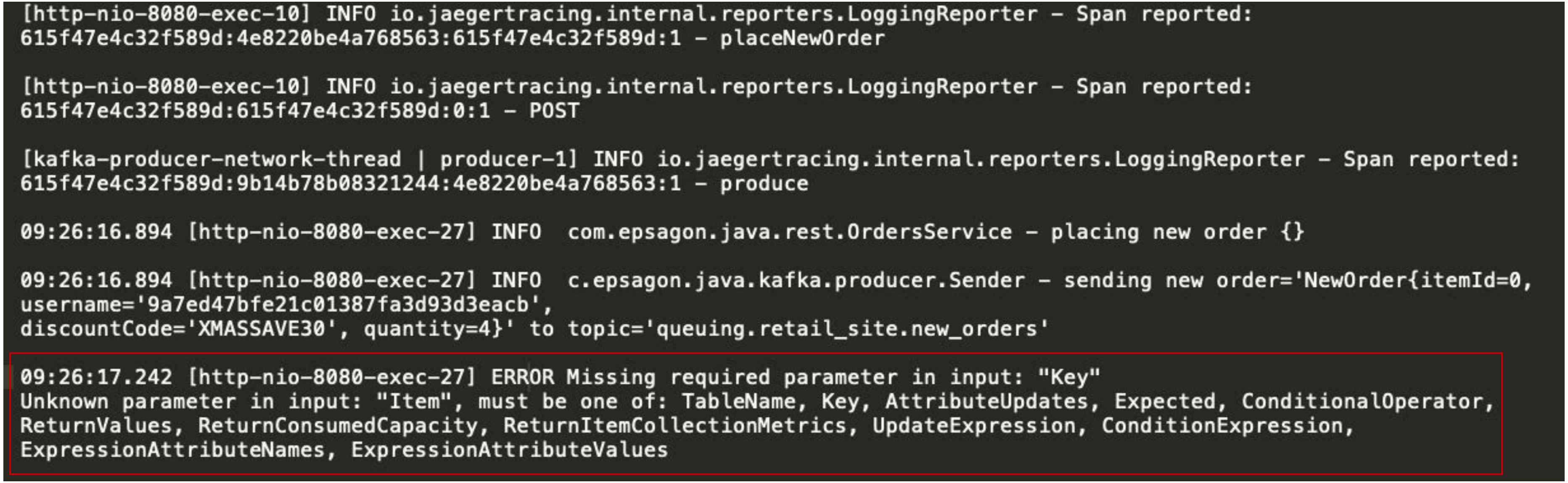
Troubleshooting

We need more debug data → logs



```
2018-11-08 20:14:50.369   redis Connecting to MASTER redis-master:6379
2018-11-08 20:14:50.370   redis MASTER <-> SLAVE sync started
2018-11-08 20:15:04.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&value=Lorem+ipsum+dolor+sit+amet%2C+consecteteur+adipiscing.+B%27duis%27+b%27mi%27+b%27a%27+b%27quam%27.+B%27eget%27+b%27ad%27+b%27a%27+ HTTP/1.1" 200
                          255
2018-11-08 20:15:04.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:15:13.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:15:13.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&value=Lorem+ipsum+dolor+sit+amet%2C+consecteteur.+B%27eros%27+b%27ve%27+b%27a%27+b%27a%27+b%27a%27+b%27curabitur%27+b%27a%27+b%27odio%27+b%27a HTTP/1.1"
                          200 255
2018-11-08 20:15:16.000   REPLCONF ACK 13902
2018-11-08 20:15:17.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:15:17.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&
                          value=Lorem+ipsum.+B%27amet%27+b%27ac%27.+B%27orci%27+b%27ut%27.+B%27pede%27+b%27eu%27+b%27ac%27+b%27quam%27+b%27a%27+b%27a%27+b%27a%27+b%27enim%27+b%27a%27.+ HTTP/1.1" 200 255
2018-11-08 20:15:19.000   REPLCONF ACK 14037
2018-11-08 20:15:20.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&value=Lorem+ipsum+dolor+sit+amet%2C+consecteteur+adipiscing.+B%27arcu%27+b%27ut%27+b%27a%27+b%27a%27+b%27a%27+b%27taciti%27+b%27a%27.+B%27 HTTP/1.1" 200
                          255
2018-11-08 20:15:20.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:15:21.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&
                          value=Lorem+ipsum+dolor.+B%27quis%27+b%27in%27+b%27a%27+b%27sem%27+b%27sed%27+b%27nibh%27+b%27amet%27+b%27a%27+b%27a%27+b%27nibh%27.+B%27nisi%27+b%27ad%27 HTTP/1.1" 200 255
2018-11-08 20:15:21.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:15:26.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:15:26.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&value=Lorem+ipsum+dolor+sit+amet.+B%27nunc%27+b%27et%27+b%27egestas%27+b%27a%27+b%27nec%27+b%27sociis%27+b%27purus%27+b%27nec%27.+B%27eget%27+ HTTP/1.1"
                          200 255
2018-11-08 20:15:34.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&value=Lorem+ipsum+dolor+sit+amet%2C+consecteteur+adipiscing+elit+b%27amet%27+b%27mi%27.+B%27ante%27+b%27in%27+b%27laoreet%27+b%27s HTTP/1.1" 200 255
2018-11-08 20:15:34.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:15:41.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&value=Lorem+ipsum+dolor.+B%27pede%27+b%27ad%27.+B%27ante%27+b%27ve%27+b%27a%27+b%27etiam%27.+B%27nunc%27+b%27eu%27+b%27a%27+b%27a%27+b%27aliquet%27+b%27
                          HTTP/1.1" 200 255
2018-11-08 20:15:41.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&
                          value=Lorem+ipsum+dolor.+B%27quis%27+b%27mi%27+b%27ut%27+b%27id%27+b%27a%27+b%27a%27+b%27nam%27.+B%27eros%27+b%27et%27+b%27a%27+b%27id%27
2018-11-08 20:15:41.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:15:41.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:15:44.543   redis Connecting to MASTER redis-master:6379
2018-11-08 20:15:44.543   redis Timeout connecting to the MASTER...
2018-11-08 20:15:44.544   redis MASTER <-> SLAVE sync started
2018-11-08 20:15:50.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&value=Lorem+ipsum+dolor+sit+amet%2C+consecteteur+adipiscing+elit+b%27quam
2018-11-08 20:15:50.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:15:51.675   redis Timeout connecting to the MASTER...
2018-11-08 20:15:51.675   redis Connecting to MASTER redis-master:6379
2018-11-08 20:15:51.676   redis MASTER <-> SLAVE sync started
2018-11-08 20:15:57.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&value=Lorem+ipsum+dolor+sit+amet%2C+consecteteur+adipiscing+elit+b%27urna
2018-11-08 20:15:57.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:16:00.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&
                          value=Lorem+ipsum+dolor+sit.+B%27amet%27+b%27et%27+b%27eu%27+b%27ve%27+b%27a%27+b%27eros%27+b%27hac%27+b%27a%27+b%27a%27+b%27a%27+b%27leo
2018-11-08 20:16:00.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:16:01.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&value=Lorem+ipsum+dolor+sit+amet.+B%27eget%27+b%27ac%27.+B%27elit%27+b%27
                          HTTP/1.1" 200 255
2018-11-08 20:16:01.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:16:09.000   apache2 192.168.65.3 - "GET / HTTP/1.1" 200 826
2018-11-08 20:16:09.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&value=Lorem+ipsum+dolor+sit+amet%2C+consecteteur+adipiscing.+B%27quis%27+
2018-11-08 20:16:11.000   apache2 192.168.65.3 - "GET /guestbook.php?cmd=set&key=messages&value=Lorem+ipsum+dolor+sit.+B%27nunc%27+b%27id%27+b%27a%27.+B%27arcu%27+
```

Java Logs



```
[http-nio-8080-exec-10] INFO io.jaegertracing.internal.reporters.LoggingReporter - Span reported:
615f47e4c32f589d:4e8220be4a768563:615f47e4c32f589d:1 - placeNewOrder

[http-nio-8080-exec-10] INFO io.jaegertracing.internal.reporters.LoggingReporter - Span reported:
615f47e4c32f589d:615f47e4c32f589d:0:1 - POST

[kafka-producer-network-thread | producer-1] INFO io.jaegertracing.internal.reporters.LoggingReporter - Span reported:
615f47e4c32f589d:9b14b78b08321244:4e8220be4a768563:1 - produce

09:26:16.894 [http-nio-8080-exec-27] INFO  com.epsagon.java.rest.OrdersService - placing new order {}

09:26:16.894 [http-nio-8080-exec-27] INFO  c.epsagon.java.kafka.producer.Sender - sending new order='NewOrder{itemId=0,
username='9a7ed47bfe21c01387fa3d93d3eacb',
discountCode='XMASSAVE30', quantity=4}' to topic='queuing.retail_site.new_orders'

09:26:17.242 [http-nio-8080-exec-27] ERROR Missing required parameter in input: "Key"
Unknown parameter in input: "Item", must be one of: TableName, Key, AttributeUpdates, Expected, ConditionalOperator,
ReturnValues, ReturnConsumedCapacity, ReturnItemCollectionMetrics, UpdateExpression, ConditionExpression,
ExpressionAttributeNames, ExpressionAttributeValues
```

Things missing?

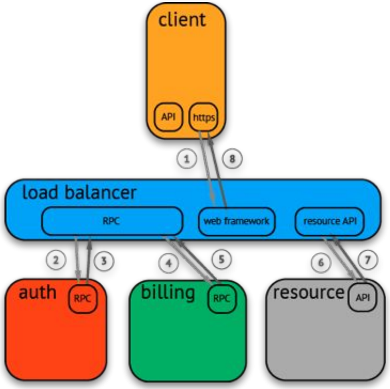
- How do we correlate between metrics and logs?
- How do we correlate data between difference services?
- How do we find the **where** when something goes wrong?

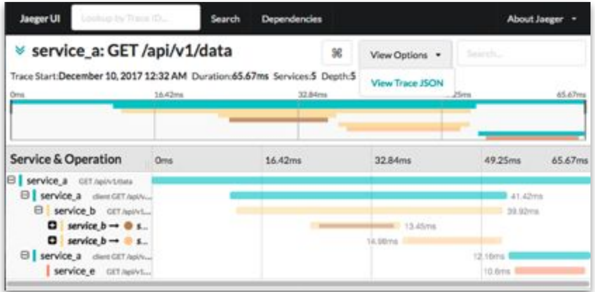
Distributed Tracing

What is Distributed Tracing?

*“A **trace** tells the story of a transaction or workflow as it propagates through a distributed system.”*

Since distributed tracing connects every request in a transaction, it allows you to know and see what’s happening to every service component and app in production





service_a: GET /api/v1/data

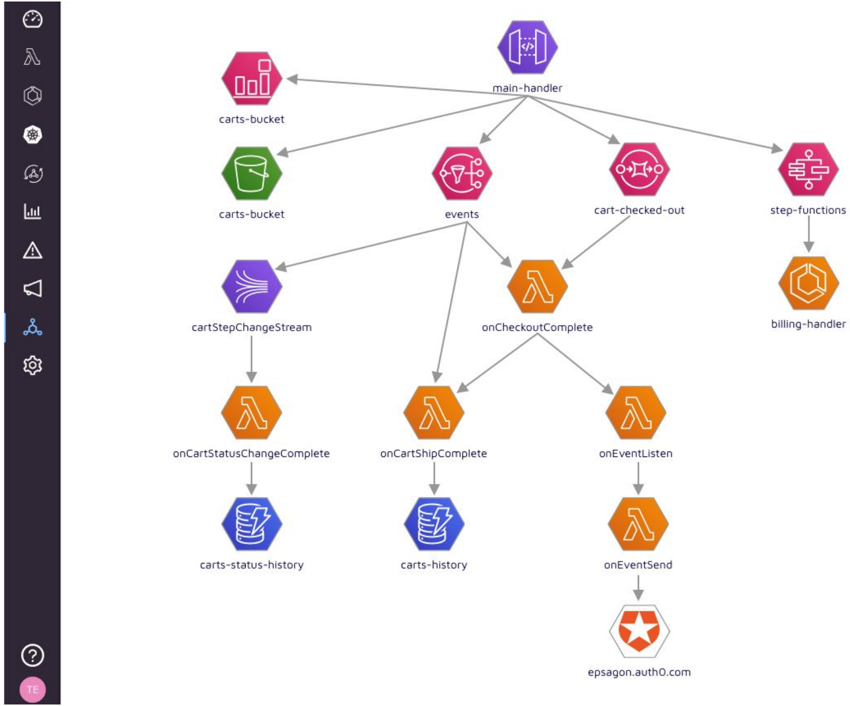
Trace Start:**December 10, 2017 12:32 AM** Duration:**65.67ms** Services:**5** Depth:**5**

```
Service & Operation          0ms     16.42ms     32.84ms     49.25ms     65.67ms
⊟ service_a  GET /api/v1/data
  ⊟ service_a  client GET /api/v...                                   41.42ms
    ⊟ service_b  GET /api/v1...                                       39.92ms
      ⊞ service_b → s...                              13.45ms
      ⊞ service_b → s...                      14.98ms
  ⊟ service_a  client GET /api/v...                           12.16ms
      service_e  GET /api/v1...                               10.6ms
```

Benefits of Distributed Tracing

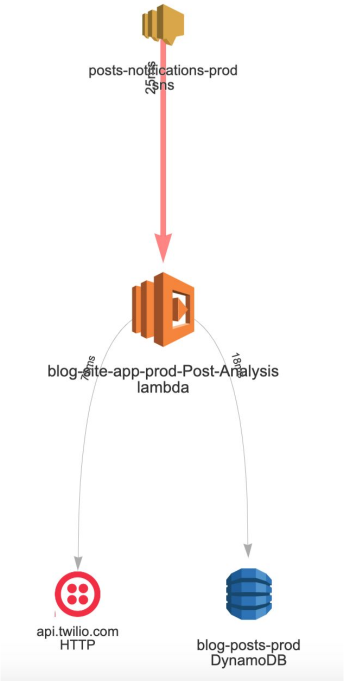
epsagon

Visualize and Understand

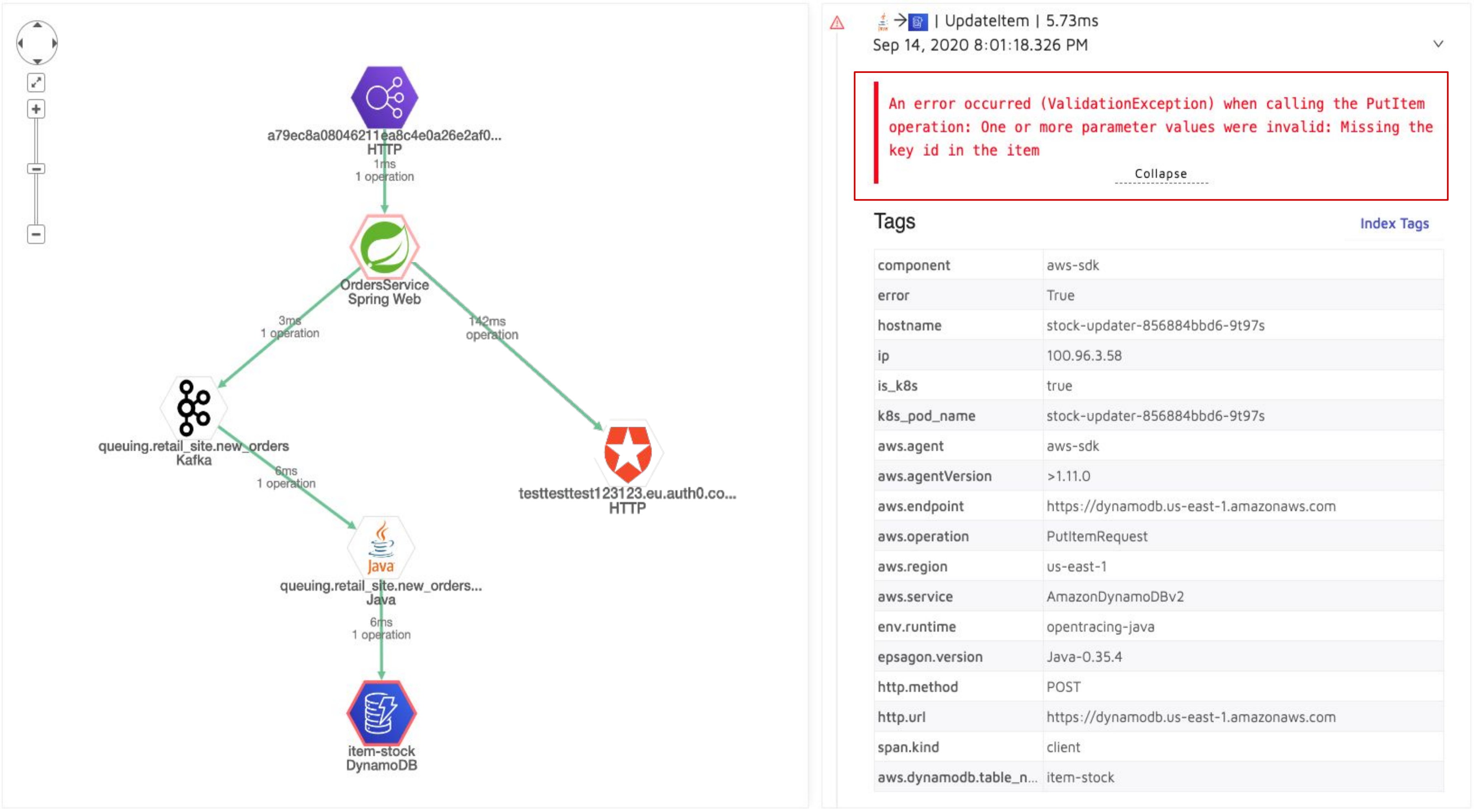


epsagon

Bring Focus to the Problems



epsagon


a79ec8a08046211ea8c4e0a26e2af0...
HTTP
1ms
1 operation
OrdersService
Spring Web
3ms
1 operation
142ms
operation
queuing.retail_site.new_orders
Kafka
6ms
1 operation
testtesttest123123.eu.auth0.co...
HTTP
queuing.retail_site.new_orders...
Java
6ms
1 operation
item-stock
DynamoDB

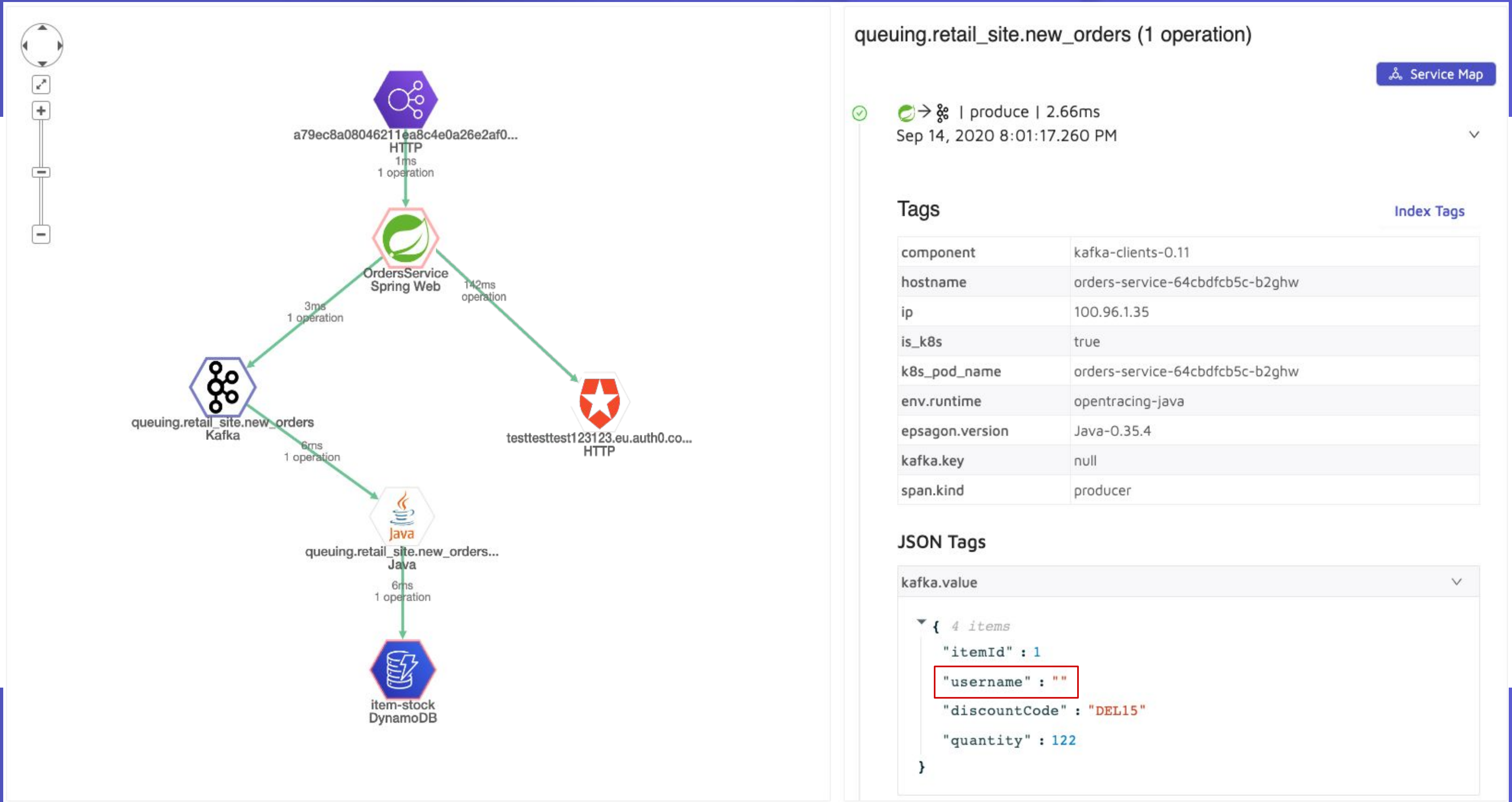
| UpdateItem | 5.73ms

Sep 14, 2020 8:01:18.326 PM

```
An error occurred (ValidationException) when calling the PutItem
operation: One or more parameter values were invalid: Missing the
key id in the item
```

Tags

component	aws-sdk
error	True
hostname	stock-updater-856884bbd6-9t97s
ip	100.96.3.58
is_k8s	true
k8s_pod_name	stock-updater-856884bbd6-9t97s
aws.agent	aws-sdk
aws.agentVersion	>1.11.0
aws.endpoint	https://dynamodb.us-east-1.amazonaws.com
aws.operation	PutItemRequest
aws.region	us-east-1
aws.service	AmazonDynamoDBv2
env.runtime	opentracing-java
epsagon.version	Java-0.35.4
http.method	POST
http.url	https://dynamodb.us-east-1.amazonaws.com
span.kind	client
aws.dynamodb.table_n...	item-stock



queuing.retail_site.new_orders (1 operation)

| produce | 2.66ms
Sep 14, 2020 8:01:17.260 PM

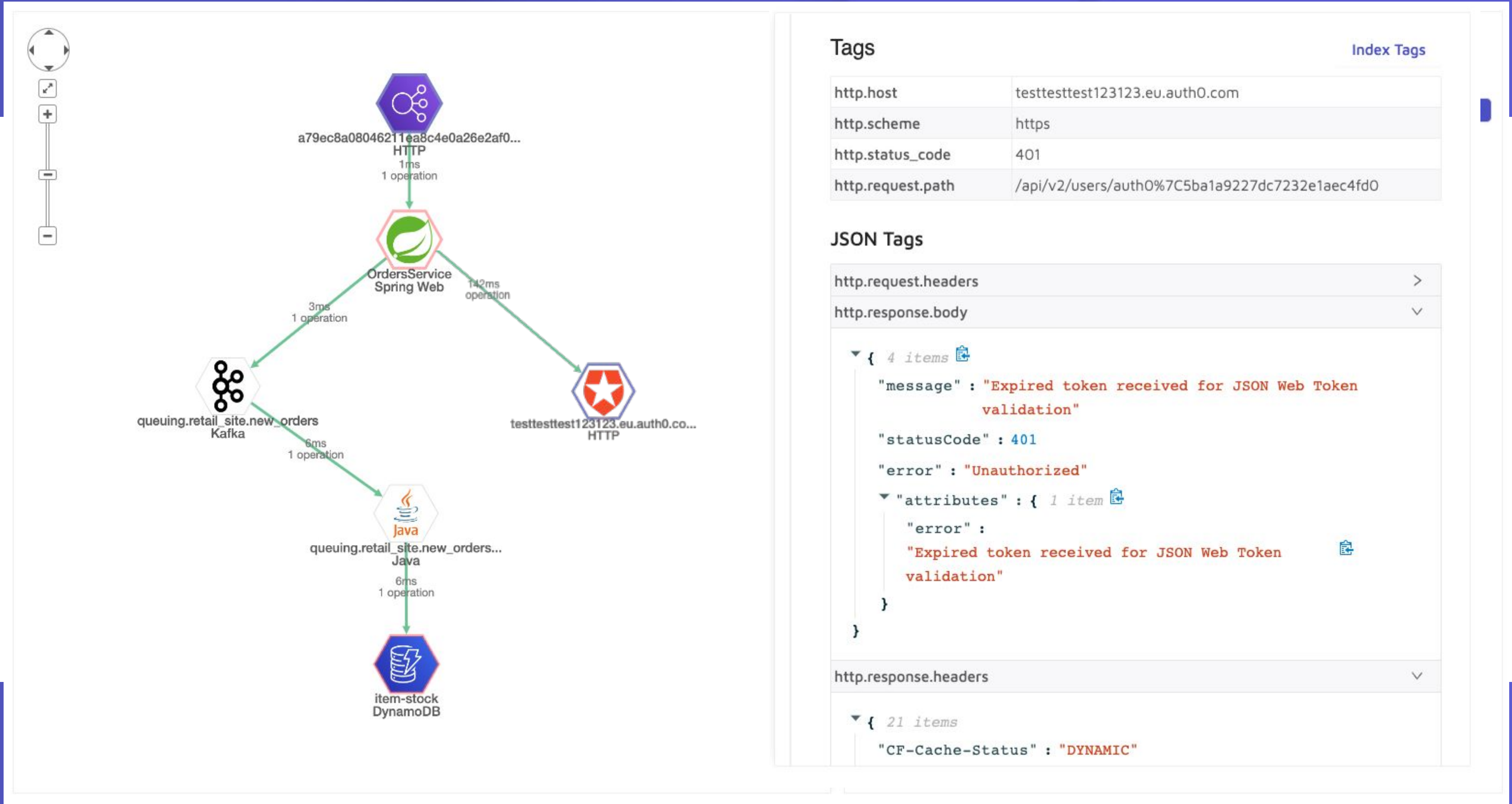
Tags

component	kafka-clients-0.11
hostname	orders-service-64cbdfcb5c-b2ghw
ip	100.96.1.35
is_k8s	true
k8s_pod_name	orders-service-64cbdfcb5c-b2ghw
env.runtime	opentracing-java
epsagon.version	Java-0.35.4
kafka.key	null
span.kind	producer

JSON Tags

kafka.value

```
{ 4 items
  "itemId" : 1
  "username" : ""
  "discountCode" : "DEL15"
  "quantity" : 122
}
```


a79ec8a08046211ea8c4e0a26e2af0...
HTTP
1ms
1 operation
OrdersService
Spring Web
3ms
1 operation
142ms
operation
queuing.retail_site.new_orders
Kafka
testtesttest123123.eu.auth0.co...
HTTP
6ms
1 operation
queuing.retail_site.new_orders...
Java
6ms
1 operation
item-stock
DynamoDB

Tags

Index Tags

http.host	testtesttest123123.eu.auth0.com
http.scheme	https
http.status_code	401
http.request.path	/api/v2/users/auth0%7C5ba1a9227dc7232e1aec4fd0

JSON Tags

http.request.headers >

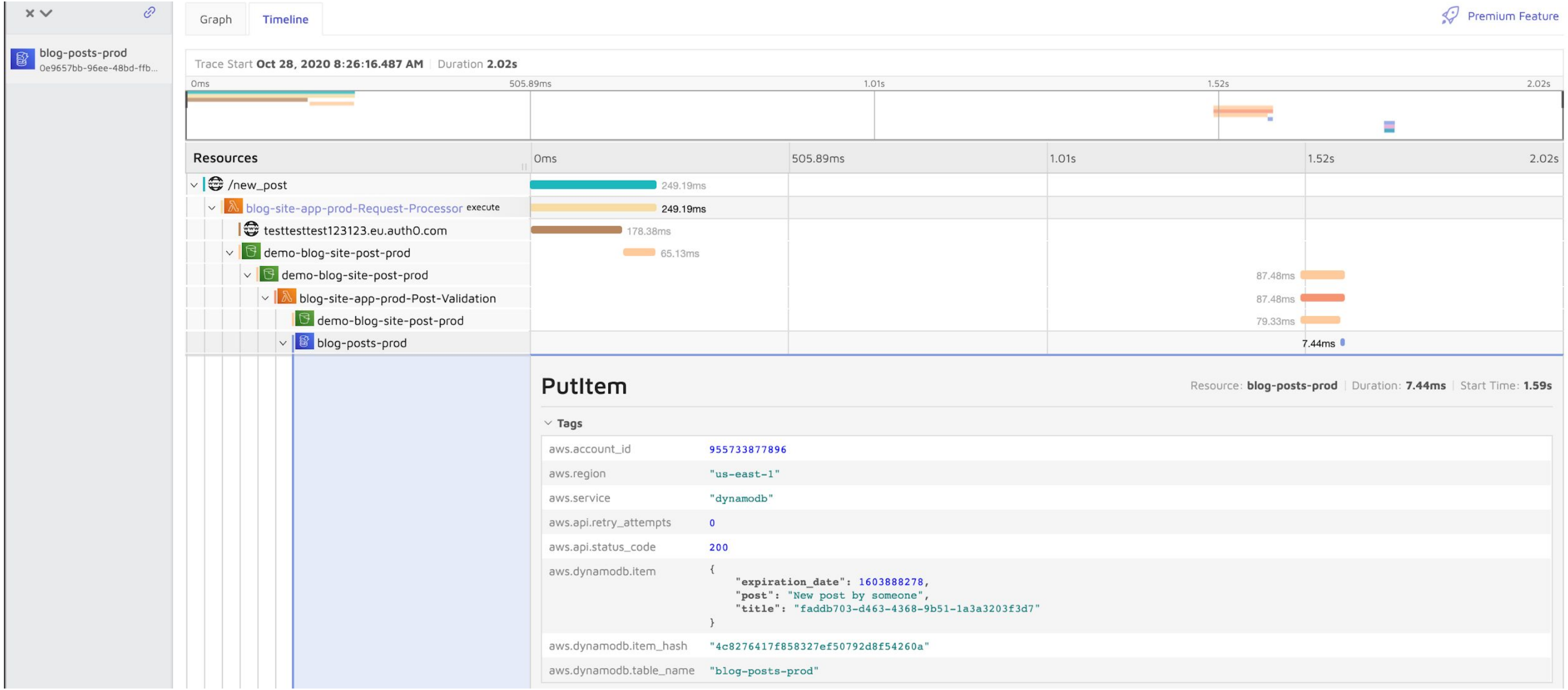
http.response.body ∨

```
{ 4 items
  "message" : "Expired token received for JSON Web Token
              validation"
  "statusCode" : 401
  "error" : "Unauthorized"
  "attributes" : { 1 item
    "error" :
    "Expired token received for JSON Web Token
    validation"
  }
}
```

http.response.headers ∨

```
{ 21 items
  "CF-Cache-Status" : "DYNAMIC"
```

Where Does Our Code Spend Time?



blog-posts-prod
0e9657bb-96ee-48bd-ffb...

Trace Start **Oct 28, 2020 8:26:16.487 AM** | Duration **2.02s**

Resources

- /new_post — 249.19ms
  - blog-site-app-prod-Request-Processor execute — 249.19ms
    - testtesttest123123.eu.auth0.com — 178.38ms
    - demo-blog-site-post-prod — 65.13ms
      - demo-blog-site-post-prod — 87.48ms
        - blog-site-app-prod-Post-Validation — 87.48ms
          - demo-blog-site-post-prod — 79.33ms
          - blog-posts-prod — 7.44ms

PutItem

Resource: **blog-posts-prod** | Duration: **7.44ms** | Start Time: **1.59s**

Tags

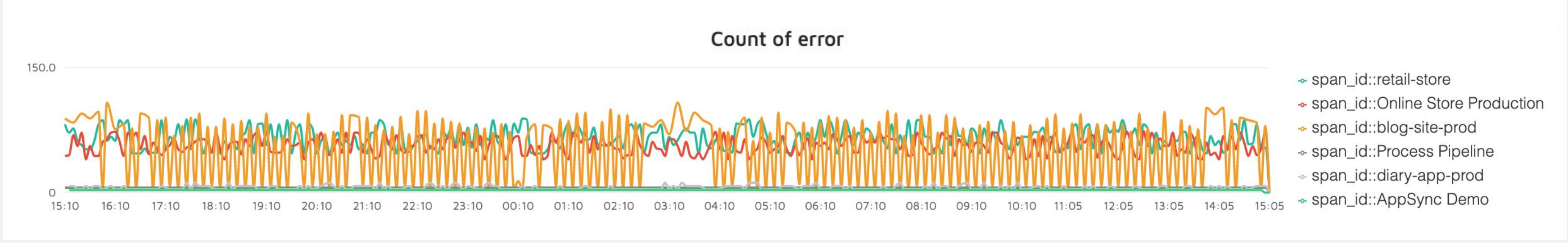
aws.account_id	955733877896
aws.region	"us-east-1"
aws.service	"dynamodb"
aws.api.retry_attempts	0
aws.api.status_code	200

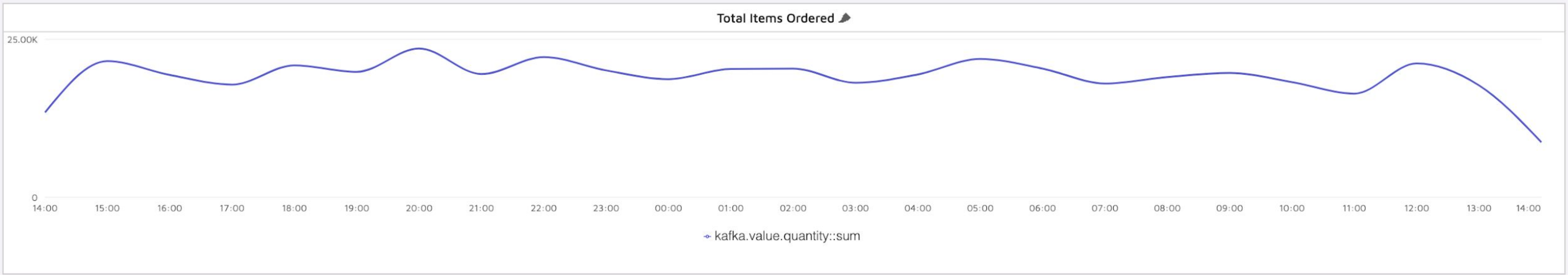
aws.dynamodb.item	{
	"expiration_date": 1603888278,
	"post": "New post by someone",
	"title": "faddb703-d463-4368-9b51-1a3a3203f3d7"
	}

aws.dynamodb.item_hash	"4c8276417f858327ef50792d8f54260a"
aws.dynamodb.table_name	"blog-posts-prod"

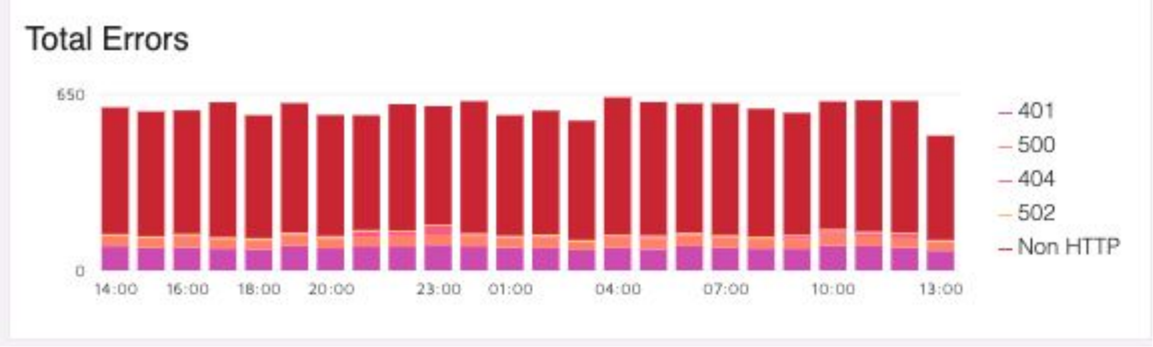
epsagon

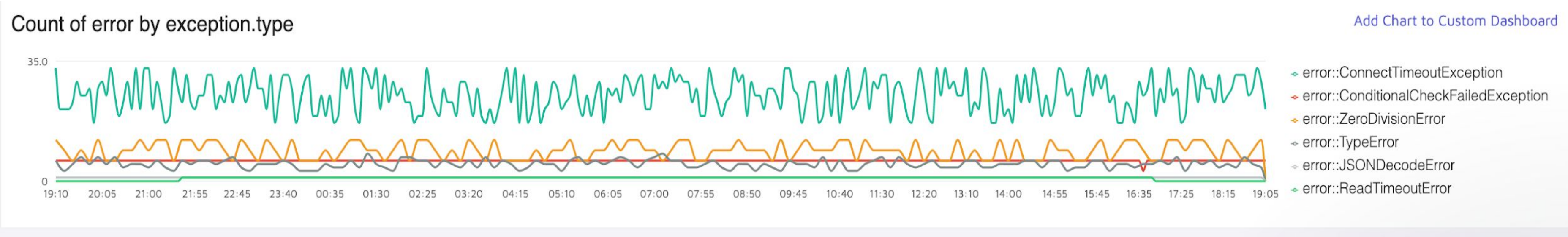
Business Insights



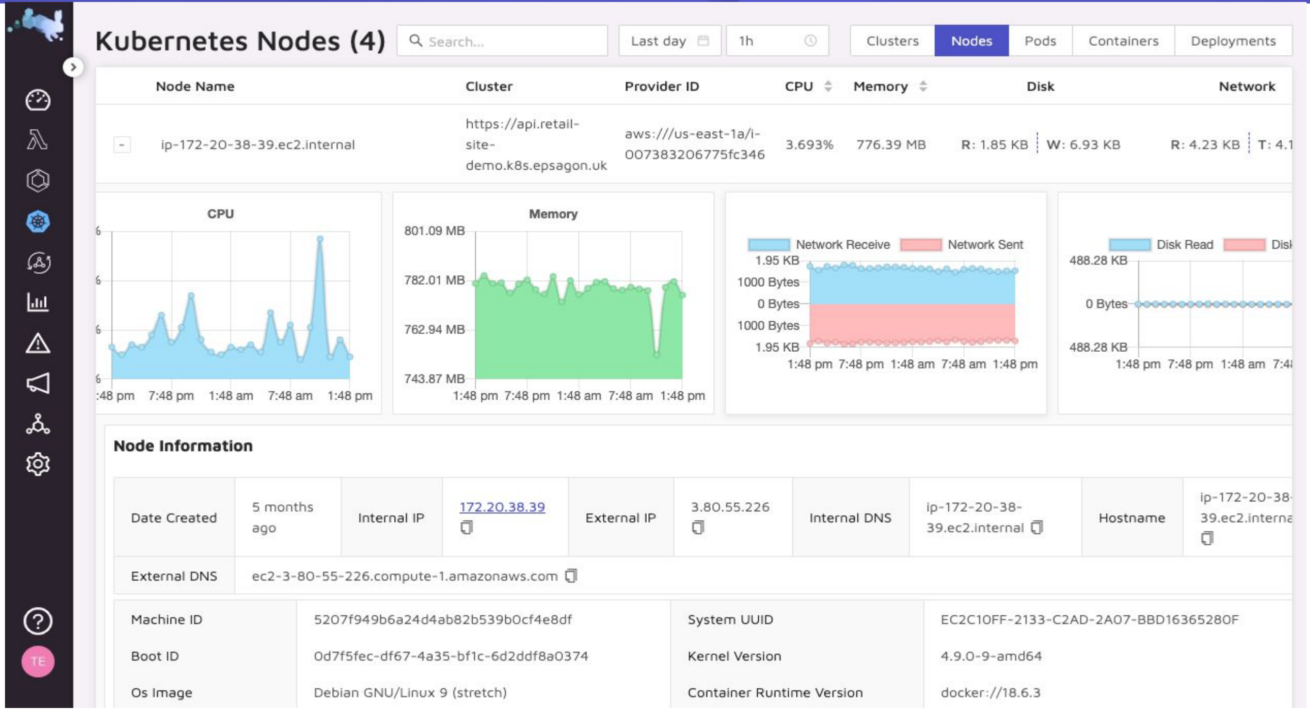


Errors, Categorized





Monitor with Trace-based Metrics and Alerts



Kubernetes Nodes (4)

Search... | Last day | 1h

Clusters | **Nodes** | Pods | Containers | Deployments

	Node Name	Cluster	Provider ID	CPU	Memory	Disk	Network
-	ip-172-20-38-39.ec2.internal	https://api.retail-site-demo.k8s.epsagon.uk	aws:///us-east-1a/i-007383206775fc346	3.693%	776.39 MB	R: 1.85 KB ¦ W: 6.93 KB	R: 4.23 KB ¦ T: 4.1

Node Information

Date Created	5 months ago	Internal IP	172.20.38.39	External IP	3.80.55.226	Internal DNS	ip-172-20-38-39.ec2.internal	Hostname	ip-172-20-38-39.ec2.interna
External DNS	ec2-3-80-55-226.compute-1.amazonaws.com								

Machine ID	5207f949b6a24d4ab82b539b0cf4e8df	System UUID	EC2C10FF-2133-C2AD-2A07-BBD16365280F
Boot ID	0d7f5fec-df67-4a35-bf1c-6d2ddf8a0374	Kernel Version	4.9.0-9-amd64
Os Image	Debian GNU/Linux 9 (stretch)	Container Runtime Version	docker://18.6.3

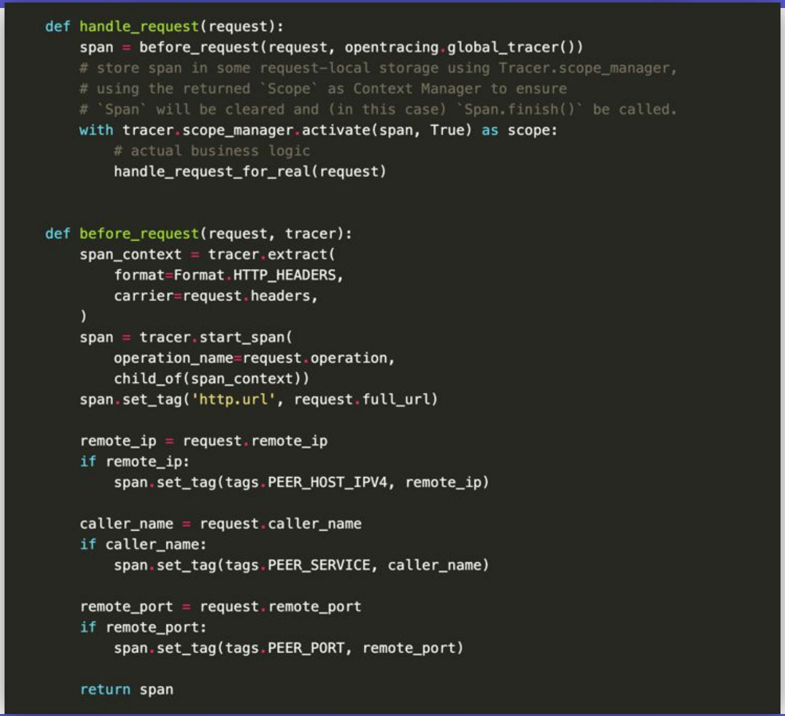
OpenTelemetry Framework, Open-source Tooling

- OpenTelemetry is a framework, not a service!
- Jaeger (Uber) and Zipkin (Twitter)
- Manual tracing requires heavy lifting: instrumentation and maintenance
- Lack visualizations, context, and tracing *through* middleware

OpenTelemetry JAEGER ZIPKIN

Generating Traces with OpenTelemetry

- **Instrument** every call (AWS-SDK, http, postgres, Spring, Flask, Express, …)
- Create a **span** for every request and response
- Add **context** to every span
- **Inject** and **Extract** IDs in relevant calls



```python
def handle_request(request):
    span = before_request(request, opentracing.global_tracer())
    # store span in some request-local storage using Tracer.scope_manager,
    # using the returned `Scope` as Context Manager to ensure
    # `Span` will be cleared and (in this case) `Span.finish()` be called.
    with tracer.scope_manager.activate(span, True) as scope:
        # actual business logic
        handle_request_for_real(request)


def before_request(request, tracer):
    span_context = tracer.extract(
        format=Format.HTTP_HEADERS,
        carrier=request.headers,
    )
    span = tracer.start_span(
        operation_name=request.operation,
        child_of(span_context))
    span.set_tag('http.url', request.full_url)

    remote_ip = request.remote_ip
    if remote_ip:
        span.set_tag(tags.PEER_HOST_IPV4, remote_ip)

    caller_name = request.caller_name
    if caller_name:
        span.set_tag(tags.PEER_SERVICE, caller_name)

    remote_port = request.remote_port
    if remote_port:
        span.set_tag(tags.PEER_PORT, remote_port)

    return span
```

The Epsagon Solution

Easy to use
5 minute setup, fully automated, no training or maintenance required

Runs anywhere
Kubernetes, ECS, containers, serverless, and more

Correlated
metrics, logs, traces with payload visibility

End-to-end
product, from monitoring to troubleshooting across Ops and Dev

Information Security Management ISO 27001 Certified

GDPR

AICPA SOC SOC 2 TYPE II CERTIFIED

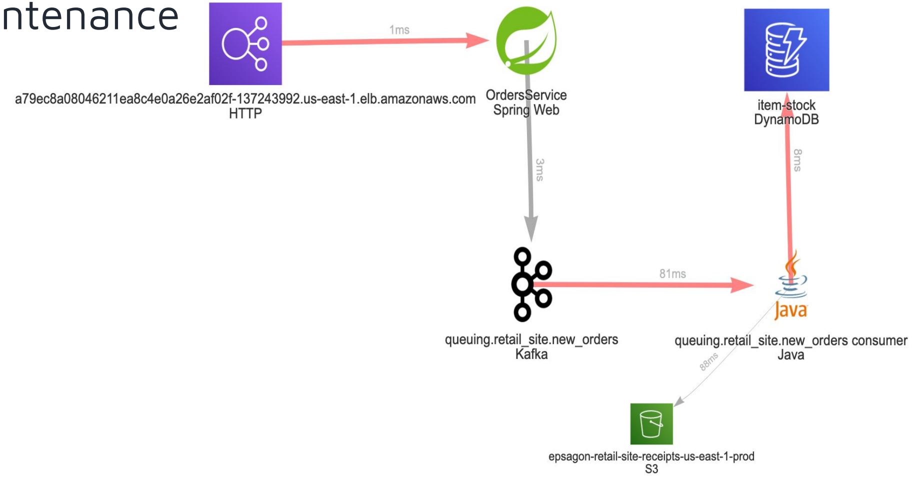
HIPAA COMPLIANT

Privacy Shield Certified

epsagon

Best Practices for Observability

- **Automated setup** and minimal maintenance
- Support **any environment** (containers, K8s, cloud, Serverless)
- Connects **every request** in a transaction
- Search and **analyze** your data
- Helps to quickly **pinpoint** problems


1ms
a79ec8a08046211ea8c4e0a26e2af02f-137243992.us-east-1.elb.amazonaws.com
HTTP
OrdersService
Spring Web
item-stock
DynamoDB
3ms
8ms
81ms
queuing.retail_site.new_orders
Kafka
Java
queuing.retail_site.new_orders consumer
Java
88ms
epsagon-retail-site-receipts-us-east-1-prod
S3

The Journey to Observability

- Identify your business goals and architecture model
- Determine your approach: DIY or managed
- Implement observability solutions
- Ensure scalability of observability strategy

Summary

- Distributed applications bring unique benefits and challenges
- Advantages of using Distributed Tracing approach
- Observability is critical to:
  - Keep track of the architecture
  - Detect performance issues and reduce MTTR
  - Reduce Ops, Dev and Opportunity costs

Be **PROACTIVE** not REACTIVE

Thank you!

chinmay@epsagon.com

To learn more and for our special offer visit:
https://epsagon.com/skilupglobal/

epsagon