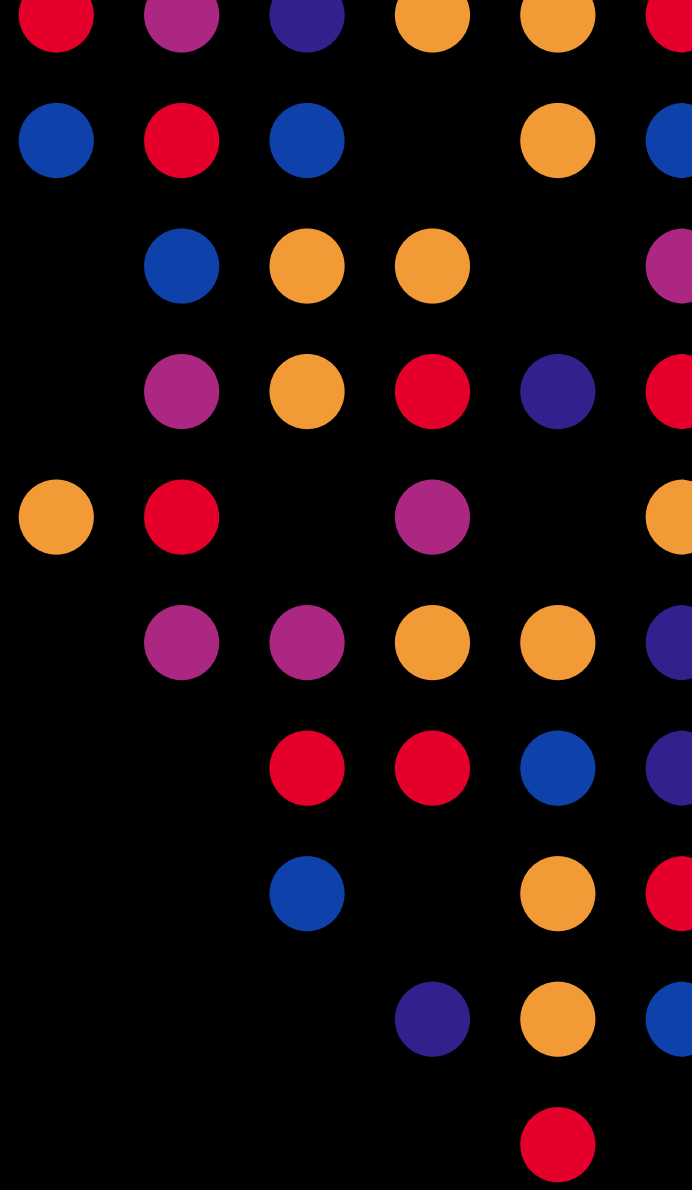
# The Anatomy of a Distributed Trace

**Dave McAllister**
Senior Open Source Technologist
NGINX



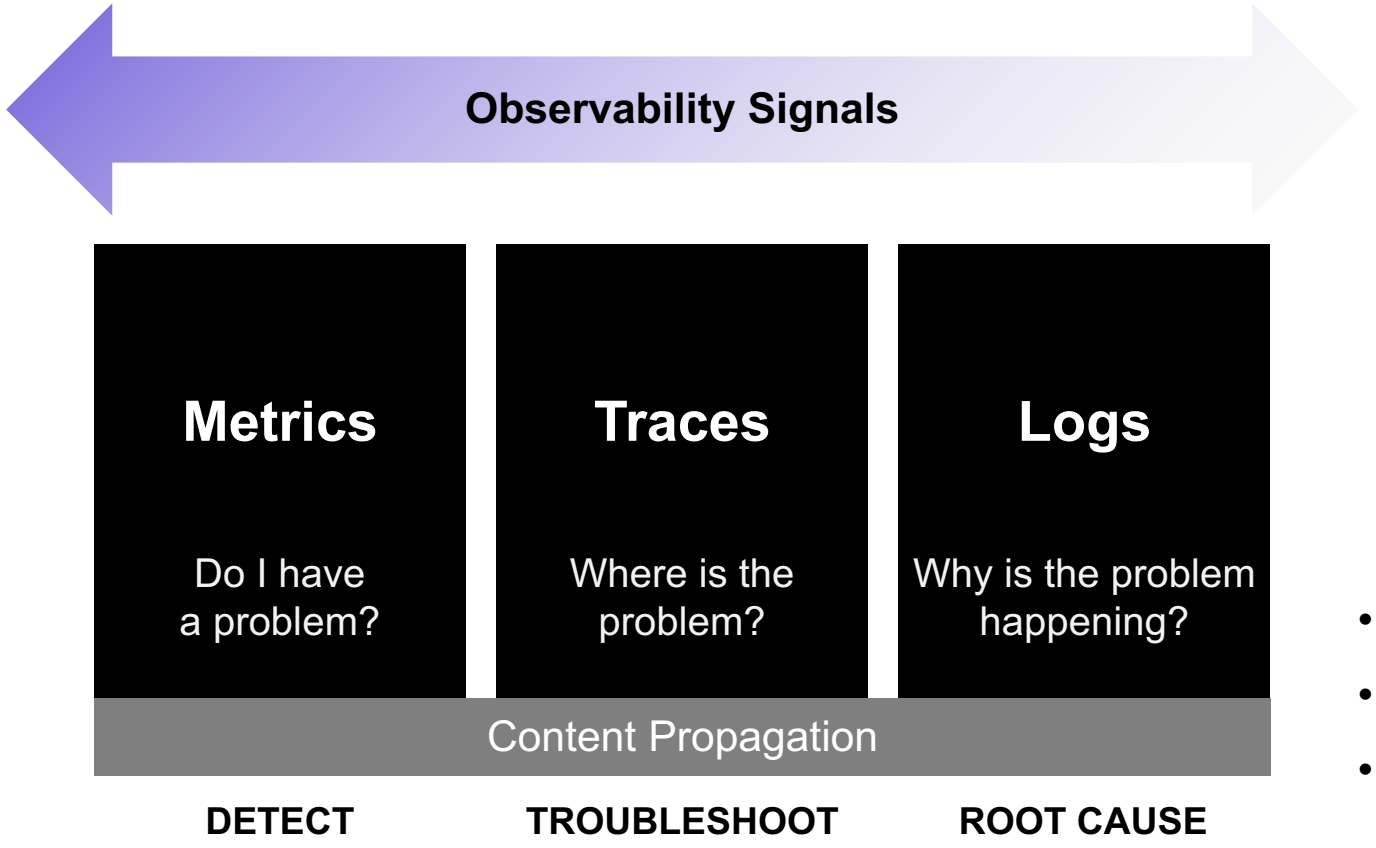
# Observability is a data problem

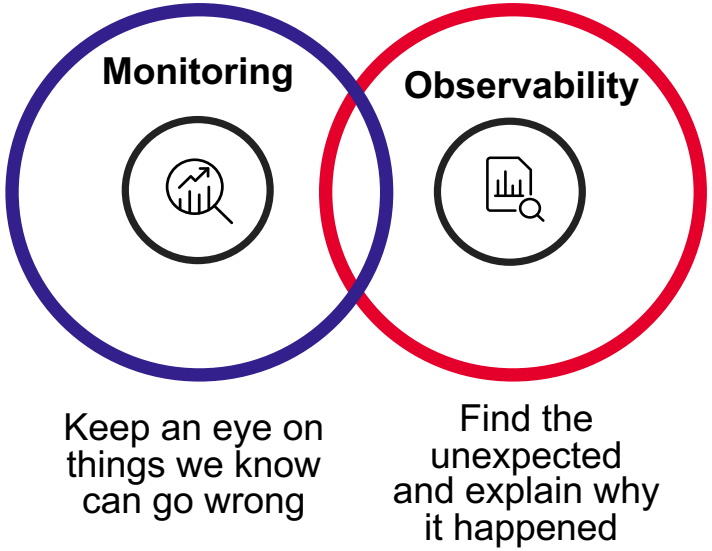
The more observable a system, the quicker we can understand why it’s acting up and fix it.

# Observability

Observability helps detect, investigate and resolve the "unknown unknowns"—fast



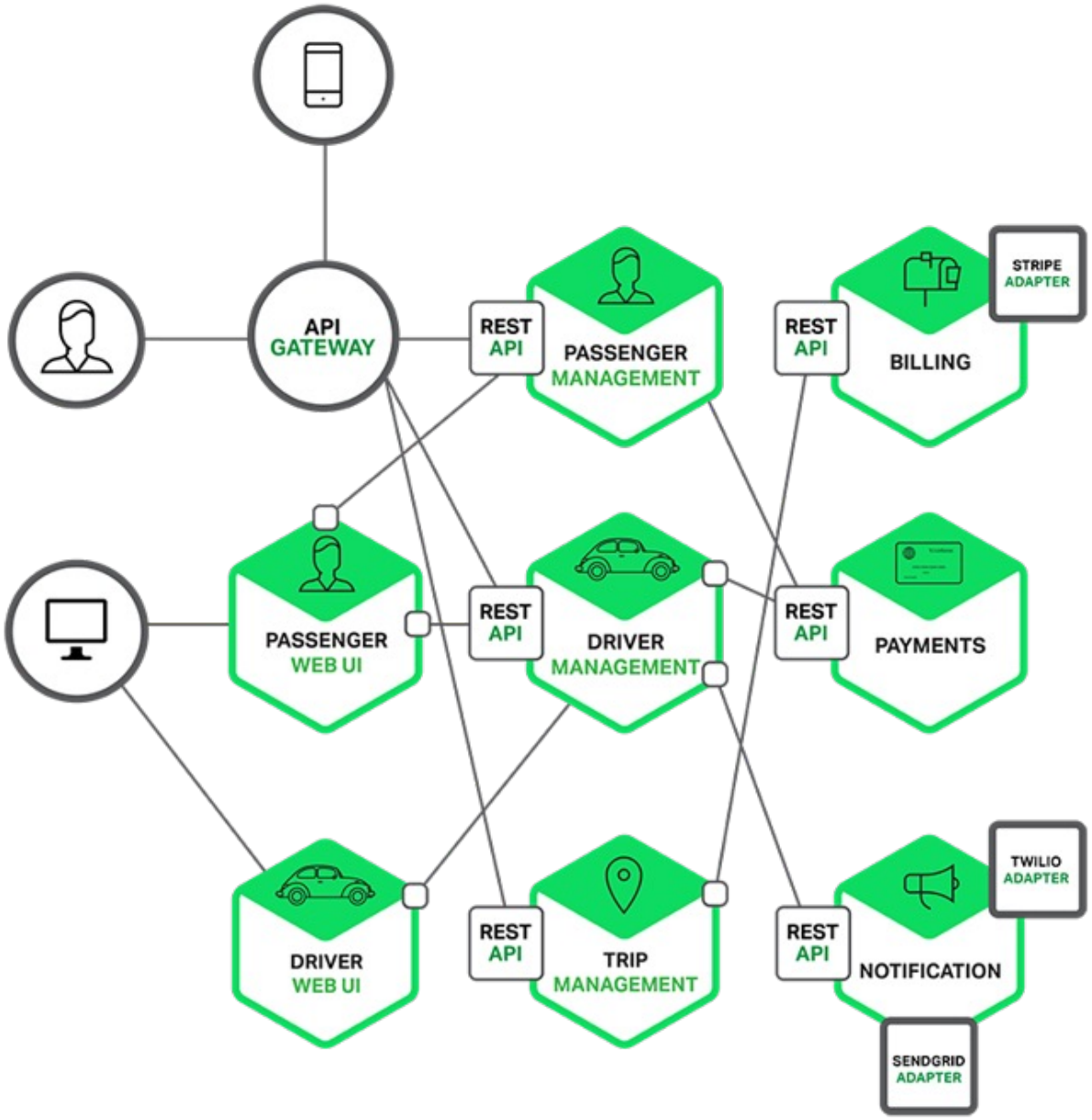

- Better visibility to the state of the system
- Precise and predictive alerting
- Reduces mean time to clue (MTTC) and mean time to resolution (MTTR)

# So why observability?

## Microservices!

A single application composed of many loosely coupled and independently deployable smaller services

- Often polyglot in nature
- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Often in cloud environments
- Organized around capabilities

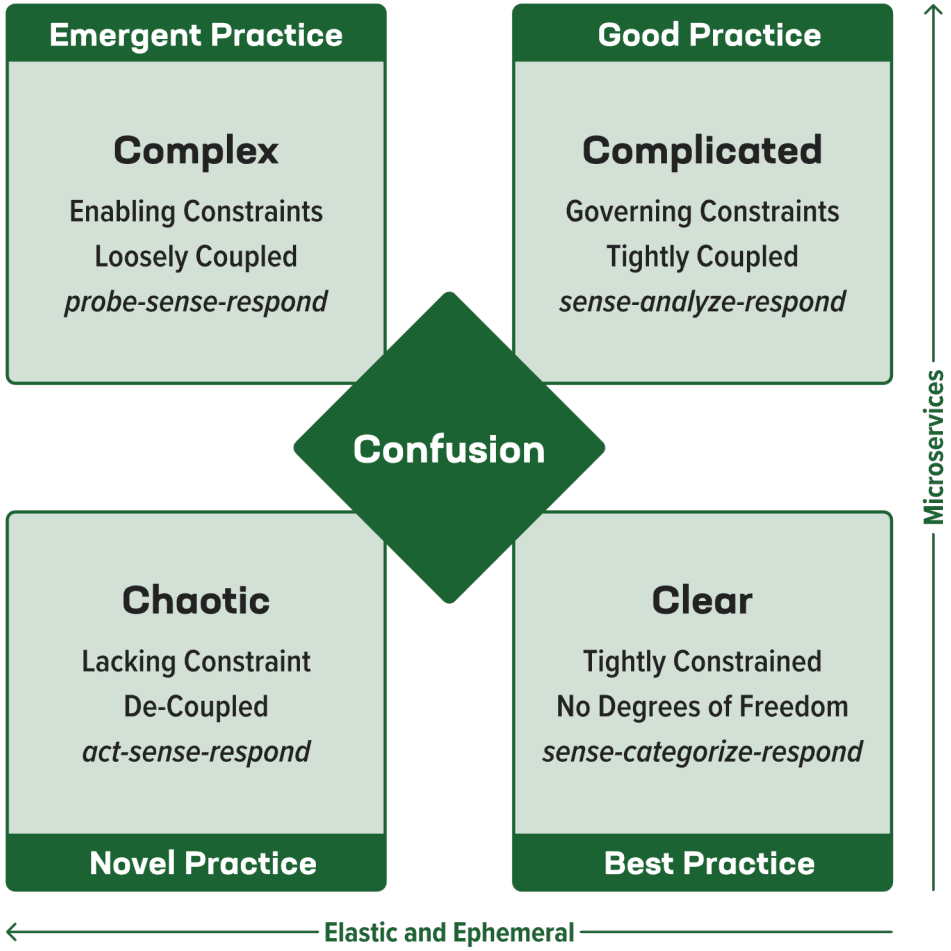


# Microservices add challenges

Especially in a cloud environment

- Microservices create complex interactions
- Failures don’t exactly repeat
- Debugging multitenancy is painful
- So much data!




Cynefin Framework

# Tracing is a data problem

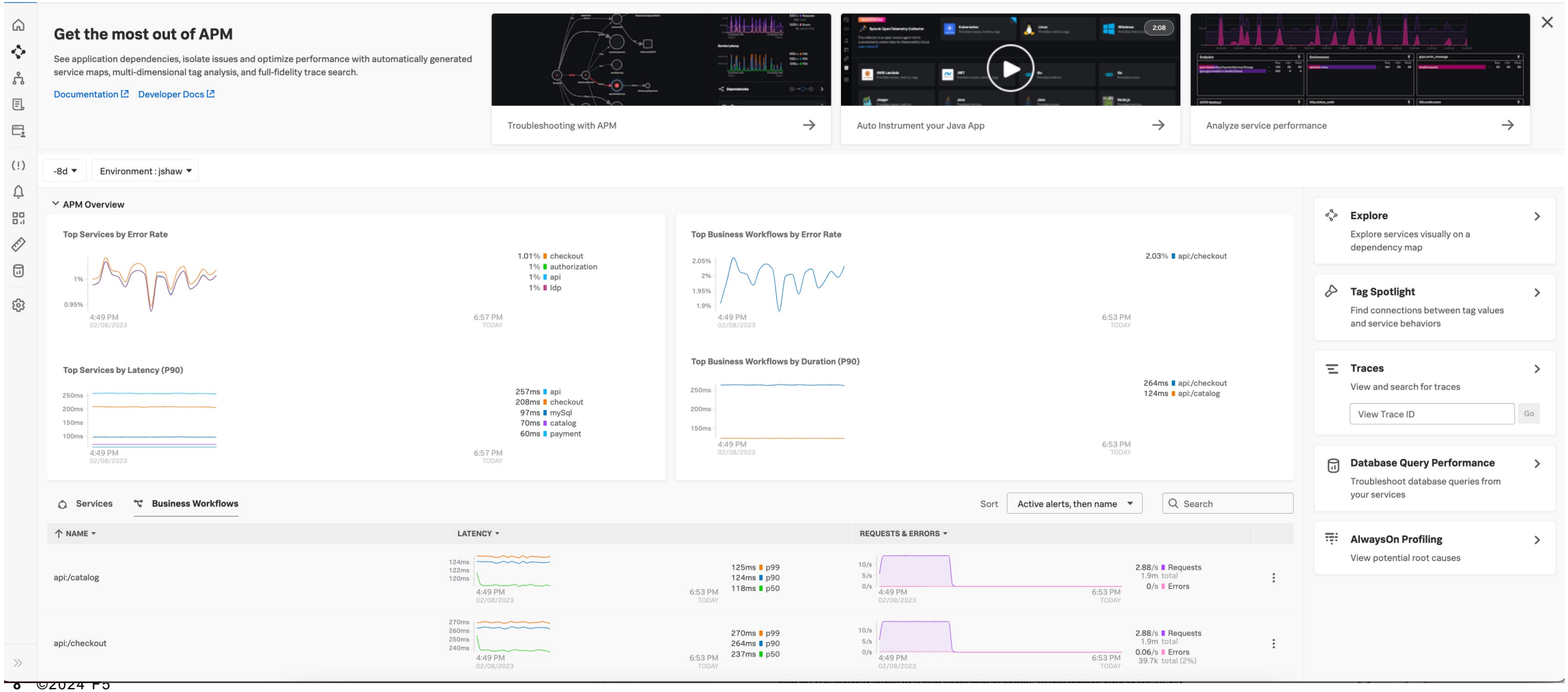
 

# What's distributed tracing good for?

- Tracks requests
- Provides actionable insights into app/user experiences
- Defines additional metrics for alerting, debugging
- Rapid MTTC, MTTR

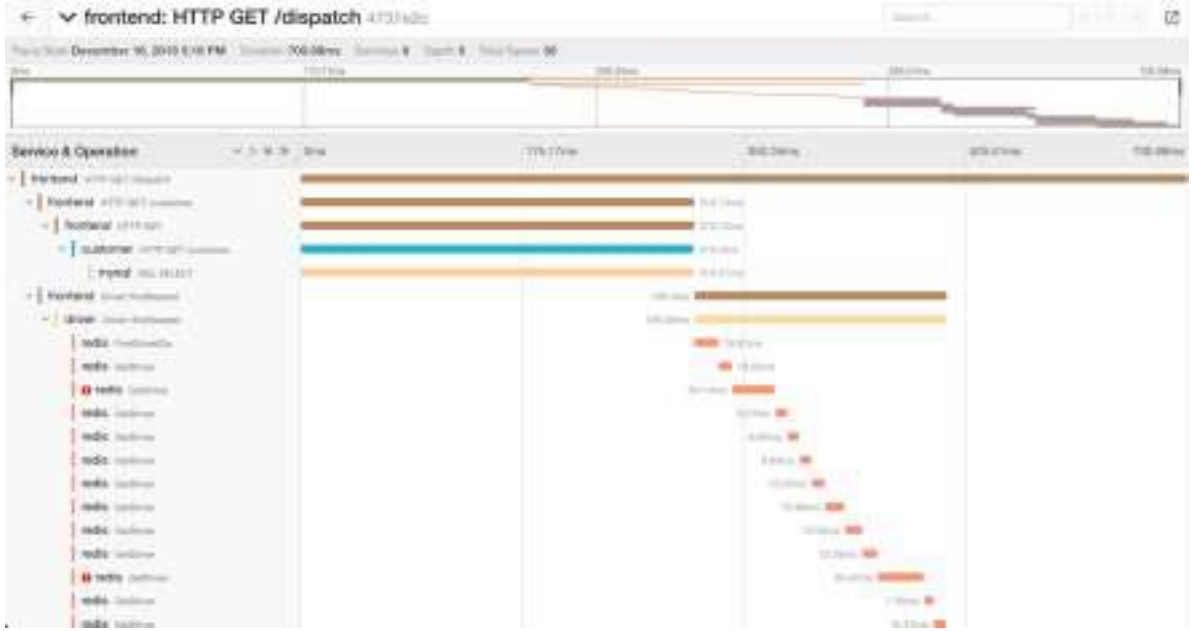
# Different models driven by observability signals

RUM, Synthetics, NPM, APM, infrastructure



# Tracing concepts

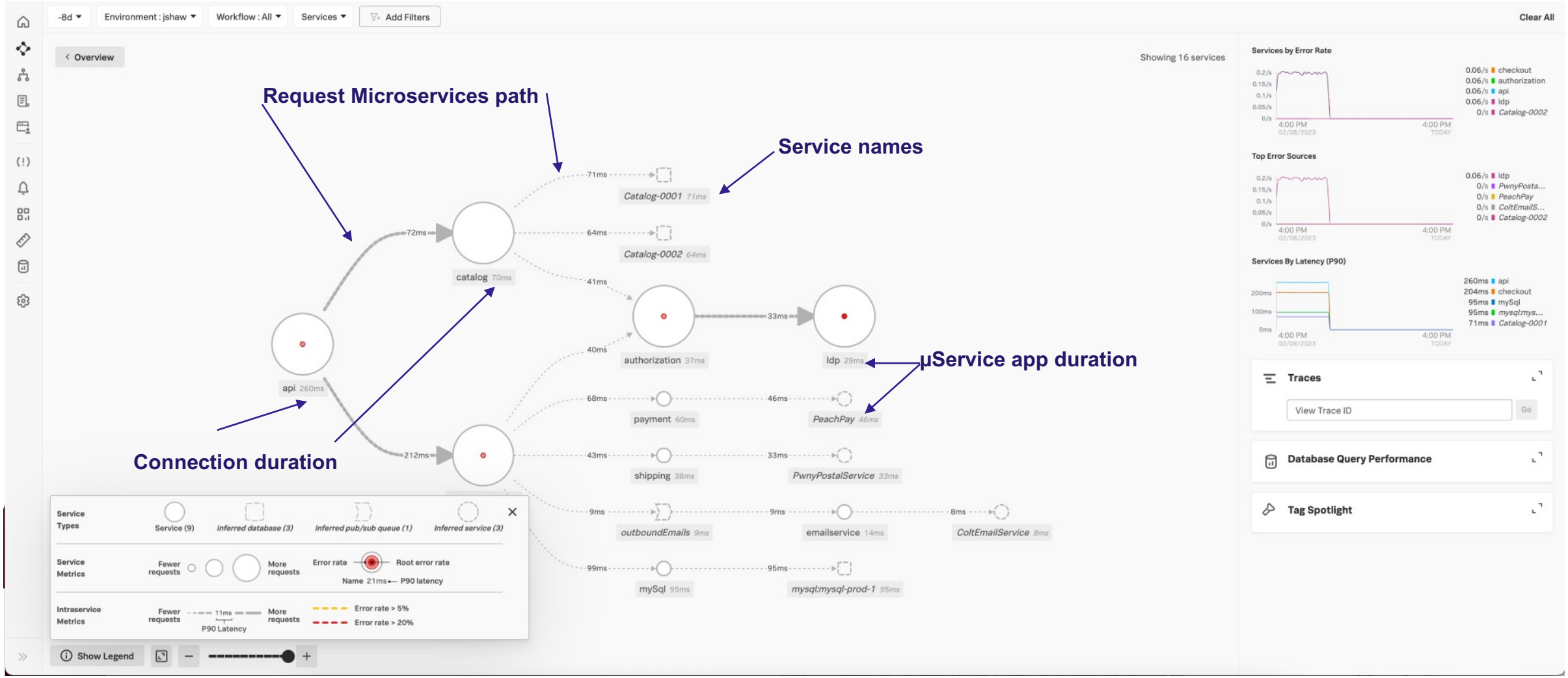
- Span
  - Represents a single unit of work in a  system
- Trace
  - Defined implicitly by its spans
- Distributed context
  - Tracing identifiers
  - Tags
  - Options that are propagated from parent to child spans





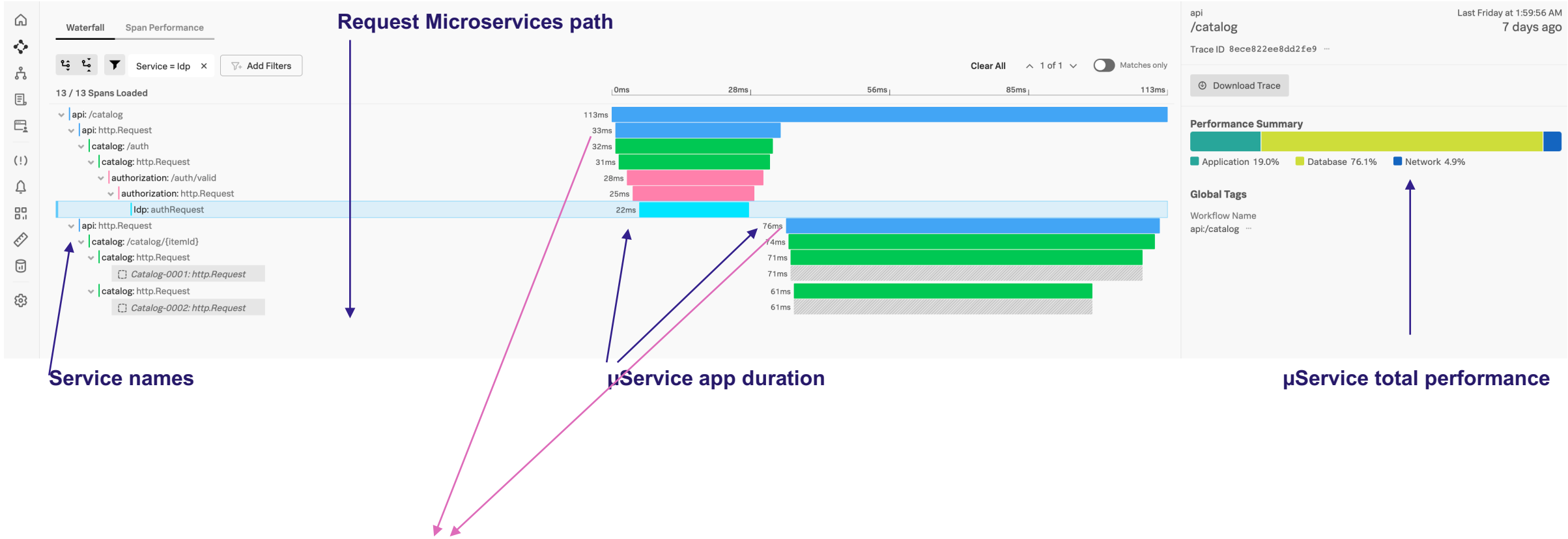

 

# Let’s look at a trace



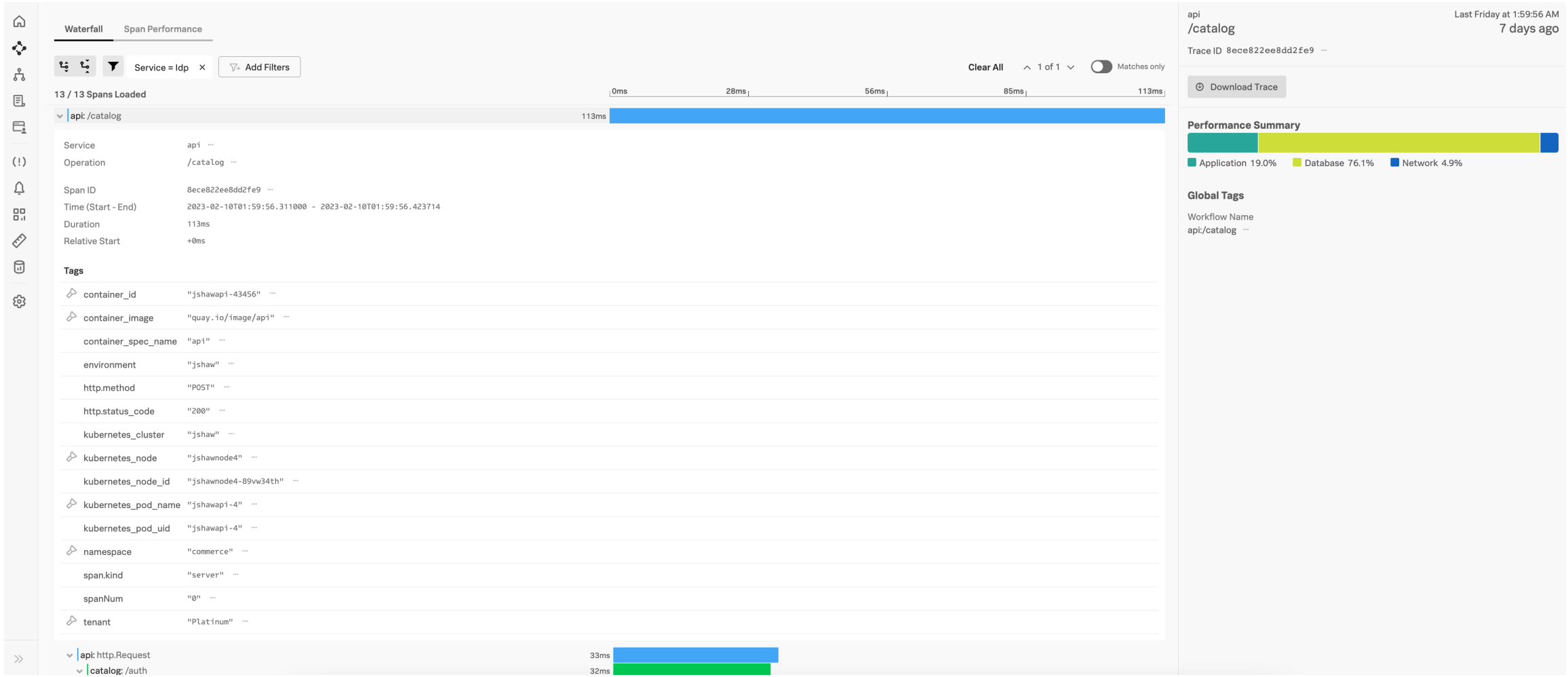

 

# A different way of looking at a trace

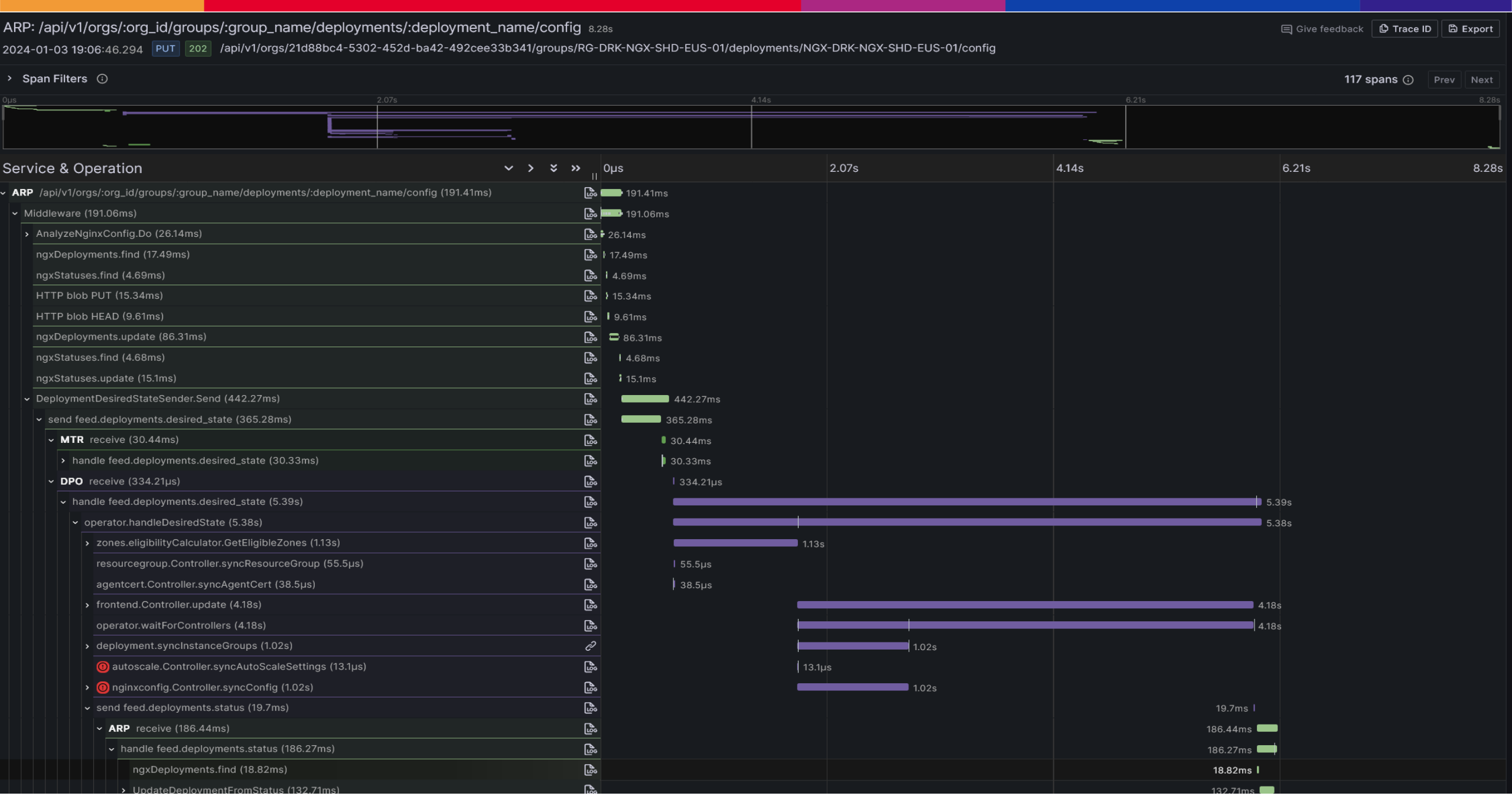



Note the two spans makes up the trace duration (almost)

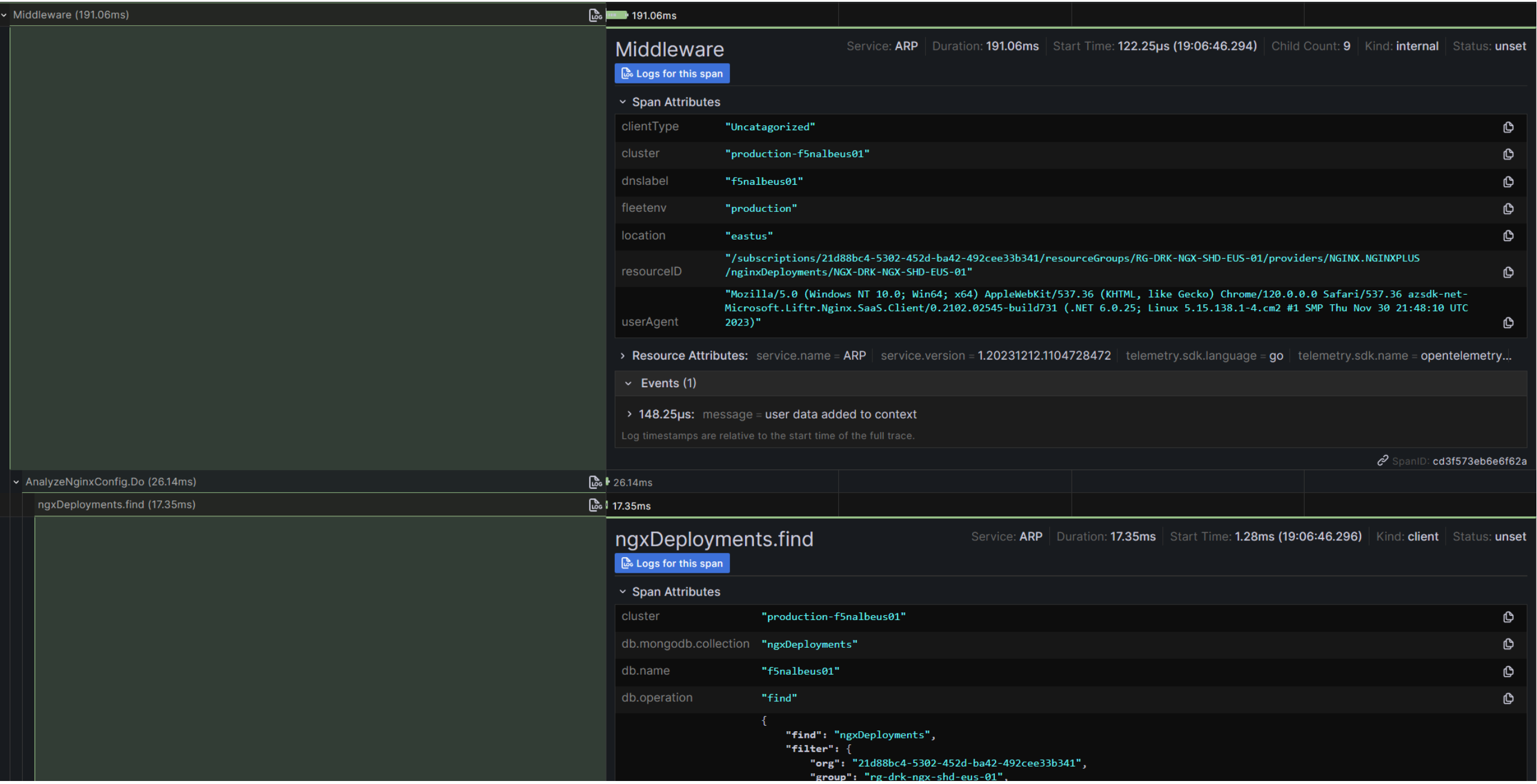
# Observability includes baggage




ARP: /api/v1/orgs/:org_id/groups/:group_name/deployments/:deployment_name/config 8.28s
Give feedback
Trace ID
Export
2024-01-03 19:06:46.294 PUT 202 /api/v1/orgs/21d88bc4-5302-452d-ba42-492cee33b341/groups/RG-DRK-NGX-SHD-EUS-01/deployments/NGX-DRK-NGX-SHD-EUS-01/config
Span Filters
117 spans
Prev
Next
0µs
2.07s
4.14s
6.21s
8.28s
Service & Operation
ARP /api/v1/orgs/:org_id/groups/:group_name/deployments/:deployment_name/config (191.41ms)
Middleware (191.06ms)
AnalyzeNginxConfig.Do (26.14ms)
ngxDeployments.find (17.49ms)
ngxStatuses.find (4.69ms)
HTTP blob PUT (15.34ms)
HTTP blob HEAD (9.61ms)
ngxDeployments.update (86.31ms)
ngxStatuses.find (4.68ms)
ngxStatuses.update (15.1ms)
DeploymentDesiredStateSender.Send (442.27ms)
send feed.deployments.desired_state (365.28ms)
MTR receive (30.44ms)
handle feed.deployments.desired_state (30.33ms)
DPO receive (334.21µs)
handle feed.deployments.desired_state (5.39s)
operator.handleDesiredState (5.38s)
zones.eligibilityCalculator.GetEligibleZones (1.13s)
resourcegroup.Controller.syncResourceGroup (55.5µs)
agentcert.Controller.syncAgentCert (38.5µs)
frontend.Controller.update (4.18s)
operator.waitForControllers (4.18s)
deployment.syncInstanceGroups (1.02s)
autoscale.Controller.syncAutoScaleSettings (13.1µs)
nginxconfig.Controller.syncConfig (1.02s)
send feed.deployments.status (19.7ms)
ARP receive (186.44ms)
handle feed.deployments.status (186.27ms)
ngxDeployments.find (18.82ms)
UpdateDeploymentFromStatus (132.71ms)
191.41ms
191.06ms
26.14ms
17.49ms
4.69ms
15.34ms
9.61ms
86.31ms
4.68ms
15.1ms
442.27ms
365.28ms
30.44ms
30.33ms
334.21µs
5.39s
5.38s
1.13s
55.5µs
38.5µs
4.18s
4.18s
1.02s
13.1µs
1.02s
19.7ms
186.44ms
186.27ms
18.82ms
132.71ms


Middleware (191.06ms)
191.06ms
Middleware
Service: ARP | Duration: 191.06ms | Start Time: 122.25µs (19:06:46.294) | Child Count: 9 | Kind: internal | Status: unset
Logs for this span
Span Attributes
clientType "Uncatagorized"
cluster "production-f5nalbeus01"
dnslabel "f5nalbeus01"
fleetenv "production"
location "eastus"
resourceID "/subscriptions/21d88bc4-5302-452d-ba42-492cee33b341/resourceGroups/RG-DRK-NGX-SHD-EUS-01/providers/NGINX.NGINXPLUS /nginxDeployments/NGX-DRK-NGX-SHD-EUS-01"
userAgent "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36 azsdk-net-Microsoft.Liftr.Nginx.SaaS.Client/0.2102.02545-build731 (.NET 6.0.25; Linux 5.15.138.1-4.cm2 #1 SMP Thu Nov 30 21:48:10 UTC 2023)"
Resource Attributes: service.name = ARP | service.version = 1.20231212.1104728472 | telemetry.sdk.language = go | telemetry.sdk.name = opentelemetry...
Events (1)
148.25µs: message = user data added to context
Log timestamps are relative to the start time of the full trace.
SpanID: cd3f573eb6e6f62a
AnalyzeNginxConfig.Do (26.14ms)
26.14ms
ngxDeployments.find (17.35ms)
17.35ms
ngxDeployments.find
Service: ARP | Duration: 17.35ms | Start Time: 1.28ms (19:06:46.296) | Kind: client | Status: unset
Logs for this span
Span Attributes
cluster "production-f5nalbeus01"
db.mongodb.collection "ngxDeployments"
db.name "f5nalbeus01"
db.operation "find"
{
"find": "ngxDeployments",
"filter": {
"org": "21d88bc4-5302-452d-ba42-492cee33b341",
"group": "rg-drk-ngx-shd-eus-01",

# What do we need for tracing?

- Generated unique IDs
- Auto-propagation
- Telemetry consolidation
- Distributed environment capable
- Standards-based agents, cloud integration
- Automated code instrumentation
- Support for developer frameworks
- Any code, any time

# Tracing API concepts – OpenTelemetry

**TracerProvider** is the entry point of the API. It provides access to Tracers.

- Stateful object holding configuration with a global provider and possibly additional ones.

**Tracer** is the class responsible for creating Spans.

- Named and optionally versioned with each instrumentation library using values guaranteed to be globally unique.
- Delegates getting active Span and marking a given Span as active to the **Context**.

**Span** is the API to trace an operation.

- Immutable **SpanContext** represents the serialized and propagated portion of a Span.

# Enabling distributed tracing

**Two basic options:**

- Traffic inspection (e.g., service mesh with context propagation)
- Code instrumentation with context propagation

**Focusing on code:**

- Add a client library dependency
- Focus on instrumenting all service-to-service communication
- Enhance spans (key-value pairs, logs)
- Add additional instrumentation (integrations, function-level, asynchronous calls)

# What does this enabling mean?

**Traces**

1. Instantiate a tracer
2. Create spans
3. Enhance spans
4. Configure SDK

**Automatic Instrumentation**

- Just add the appropriate files to the app. This is language dependent

**Manual Instrumentation**

- Import the OTel API and SDK
- Configure the API
- Configure the SDK
- Create your traces
- Create your metrics
- Export your data

# What problems does tracing clarify?

## A tiny subset of answers

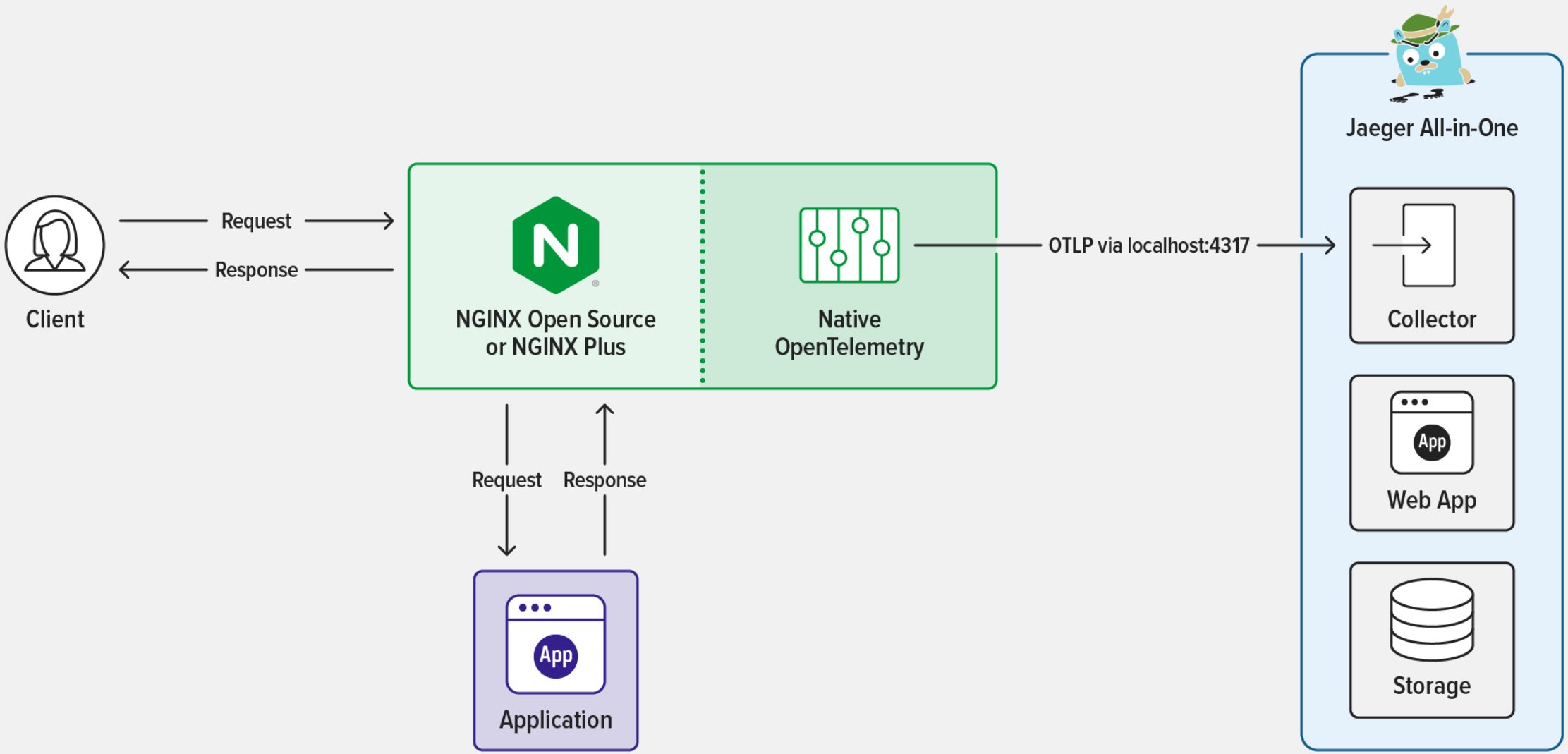
- Performance issues
- Mean time to resolution and detection
- Context for metrics and logs
- Connections between infrastructure and applications

# So Why a Native OpenTelemetry Module for NGINX?

nginxinc/nginx-otel (github.com)

- Enables NGINX to send to an Otel collector
  - Fully supports W3C trace context
  - Supports OTLP and gRPC protocols
- Performance
  - Current community modules reduce request processing by ~50%
  - Native module ~10%
- Setup and config are inline with the NGINX application configuration
- Allows dynamic control of trace parameters using cookies, tokens and/or variables
- Prebuilt packages are available, including RedHat and derivatives, Debian, Ubuntu and derivatives


Client
Request
Response
NGINX Open Source
or NGINX Plus
Native
OpenTelemetry
OTLP via localhost:4317
Request
Response
App
Application
Jaeger All-in-One
Collector
App
Web App
Storage

```
load_module modules/ngx_otel_module.so;

events {
}

http {

    otel_exporter {
        endpoint localhost:4317;
    }

    server {
        listen 127.0.0.1:8080;

        location / {
            otel_trace         on;
            otel_trace_context inject;

            proxy_pass http://backend;
        }
    }
}
```

# Example Configuration

- Otel_exporter – specifies Otel data export parameters
  - Endpoint – address to accept telemetry data
  - Interval – interval (max) between exports (5s)
  - Batch_size – max spans sent in one batch per worker (512)
  - Batch_count – number of pending batches per worker (4)
- Otel_trace – enables or disables tracing
  - Can be a variable
- Otel_trace_context – propagation directives
  - Extract | inject | propagate | ignore

# Automatically add span attributes

- http.method
- http.target
- http.route
- http.scheme
- http.flavor
- http.user_agent
- http.request_content_length
- http.response_content_length
- http.status_code
- net.host.name
- net.host.port
- net.sock.peer.addr
- net.sock.peer.port

# Simple tracing example – all HTTP requests

```
http {
    otel_exporter {
        endpoint localhost:4317;
    }

    otel_trace on;

    server {
        location / {
            proxy_pass http://backend;
        }
    }
}
```

# Parent-based tracing – inherit and record only if parent is sampled

```
http {
    server {
        location / {
            otel_trace $otel_parent_sampled;
            otel_trace_context propagate;

            proxy_pass http://backend;
        }
    }
}
```

# Ratio-based – example of percent of traffic

```
http {
    # trace 10% of requests
    split_clients $otel_trace_id $ratio_sampler {
        10%     on;
        *       off;
    }

    # or we can trace 10% of user sessions
    split_clients $cookie_sessionid $session_sampler {
        10%     on;
        *       off;
    }

    server {
        location / {
            otel_trace $ratio_sampler;
            otel_trace_context inject;

            proxy_pass http://backend;
        }
    }
}
```

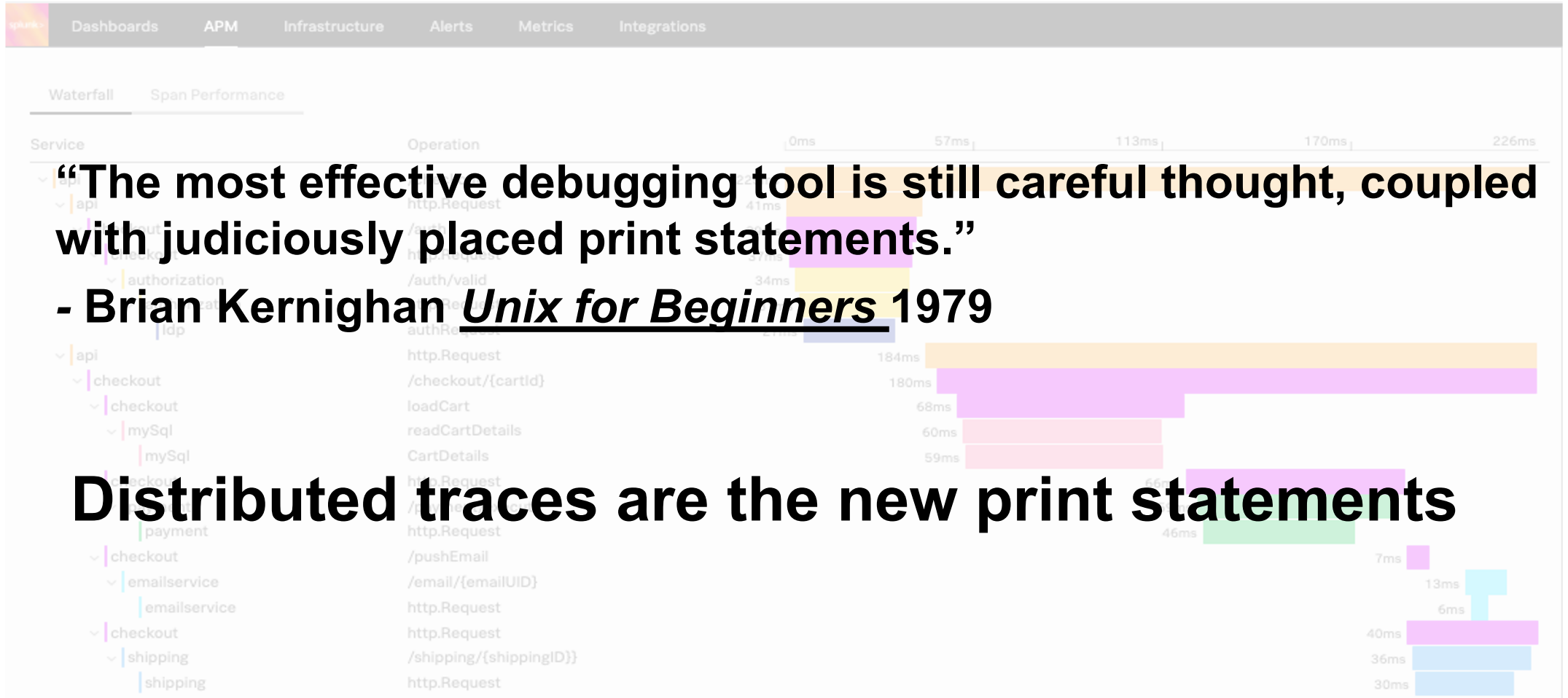
 

# Distributed tracing summary

- Gives insights into the app and its infrastructure
- Requires effort to return value
  - The “win” is when the project is already OTel instrumented
- Is a good “user happiness” proxy
- Does not magically solve your issues

# Closing thought




**“The most effective debugging tool is still careful thought, coupled with judiciously placed print statements.”**

**- Brian Kernighan *Unix for Beginners* 1979**

**Distributed traces are the new print statements**

Odysseus is in orbit, will attempt to touch down at 5:30pm Eastern time

# Thanks!

Linkedin: in/davemc

Slides on GitHub