HELLO!

OpenInfra SUMMIT › VANCOUVER '23

ANATOMY OF A DISTRIBUTED TRACE

Dave McAllister

NGINX

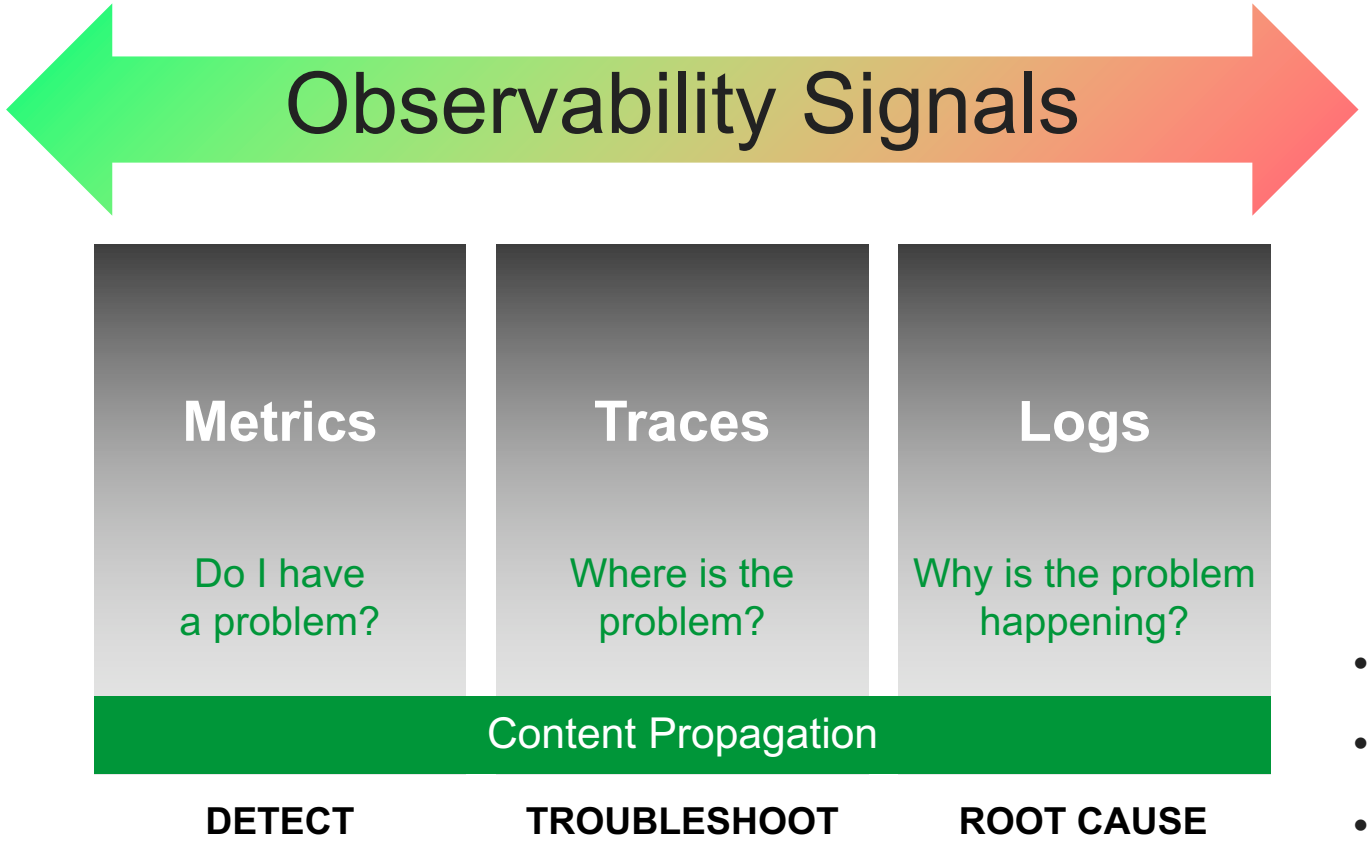
Tracing is a data problem

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



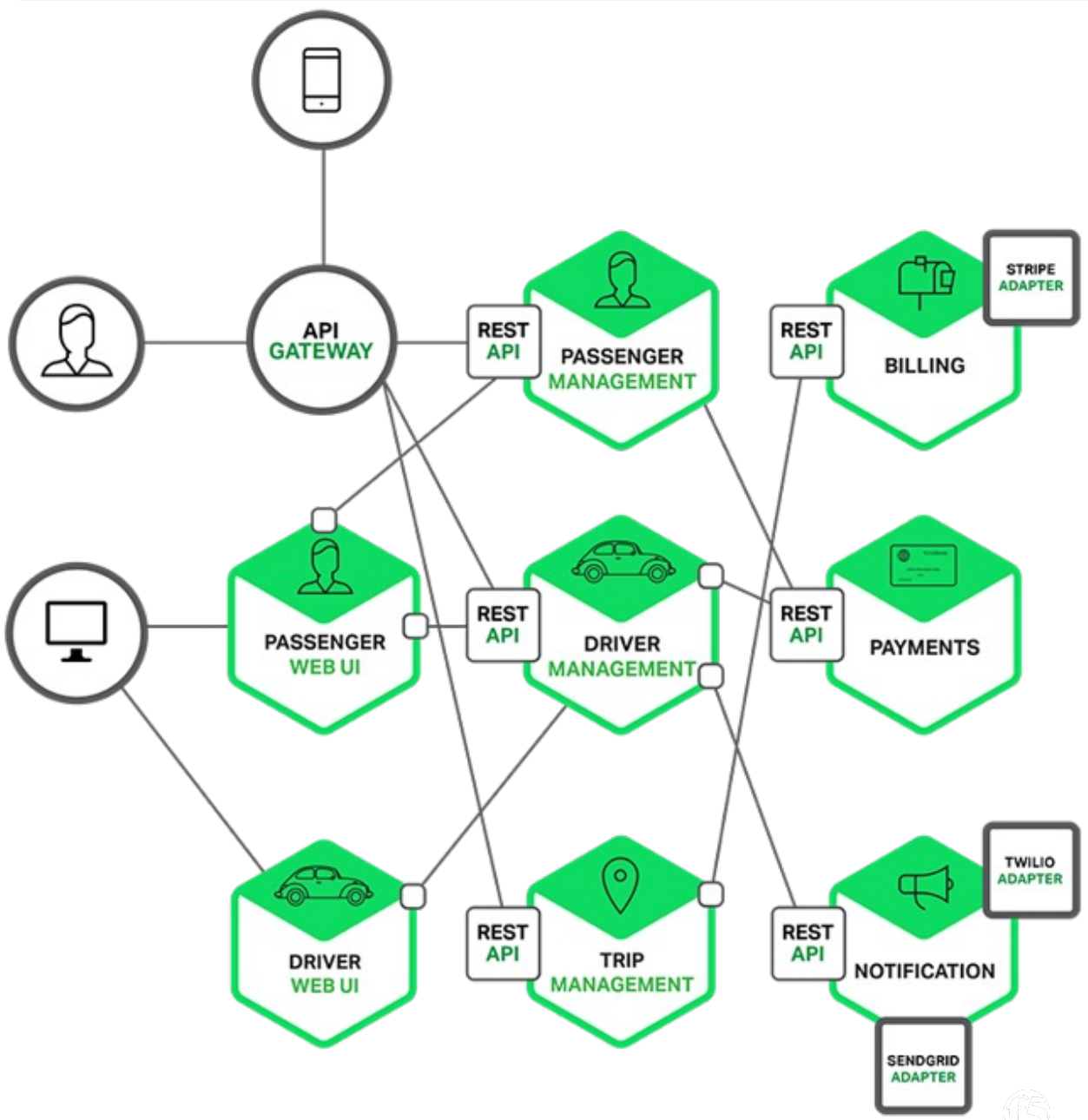
- Better visibility to the state of the system
- Precise and predictive alerting
- Reduces Mean Time to Clue (MTTC) and Mean Time to Resolution (MTTR)

So Why Observability?

Microservices!

Single application composed of many loosely coupled and independently deployable smaller services

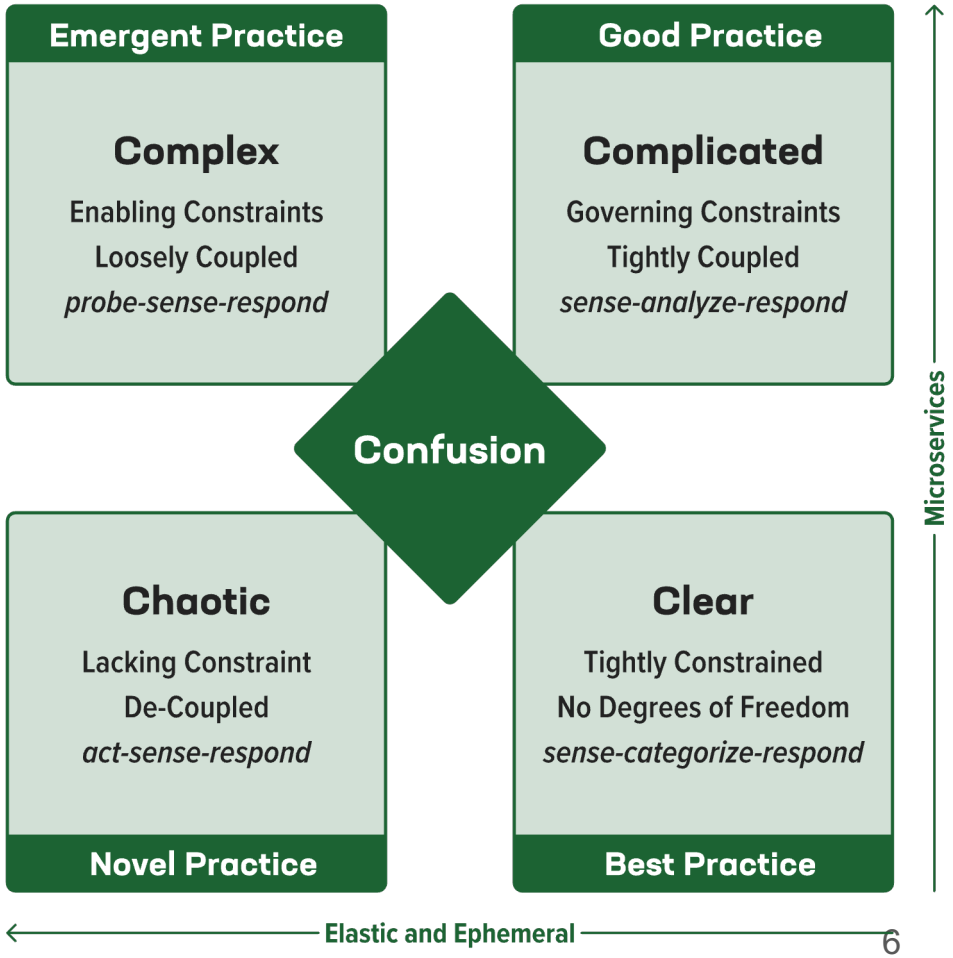
- Often polyglot in nature
- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Often in Cloud environments
- Organized around business capabilities
- Each potentially owned by a small team



But They Add Challenges

Especially when we consider this in a cloud

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



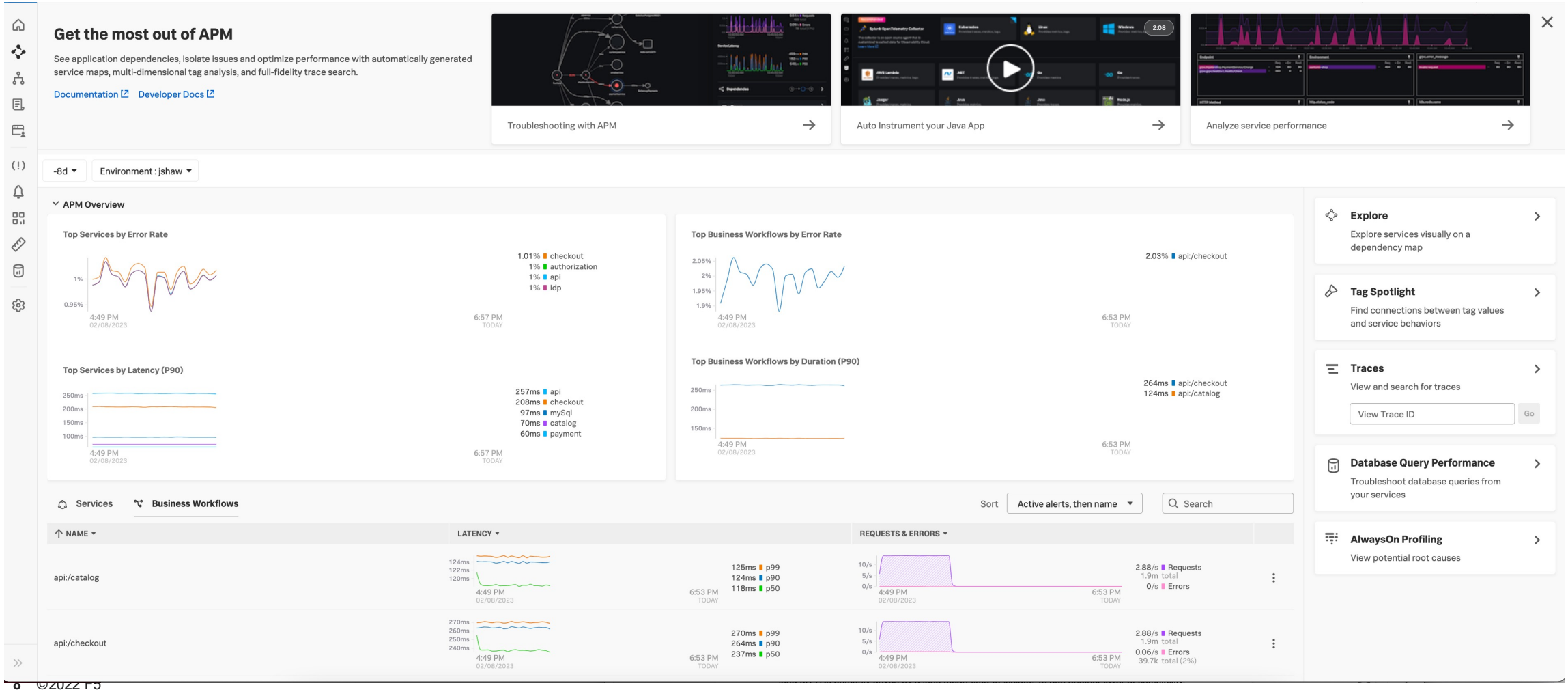
Cynefin Framework

So what's tracing good for?

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

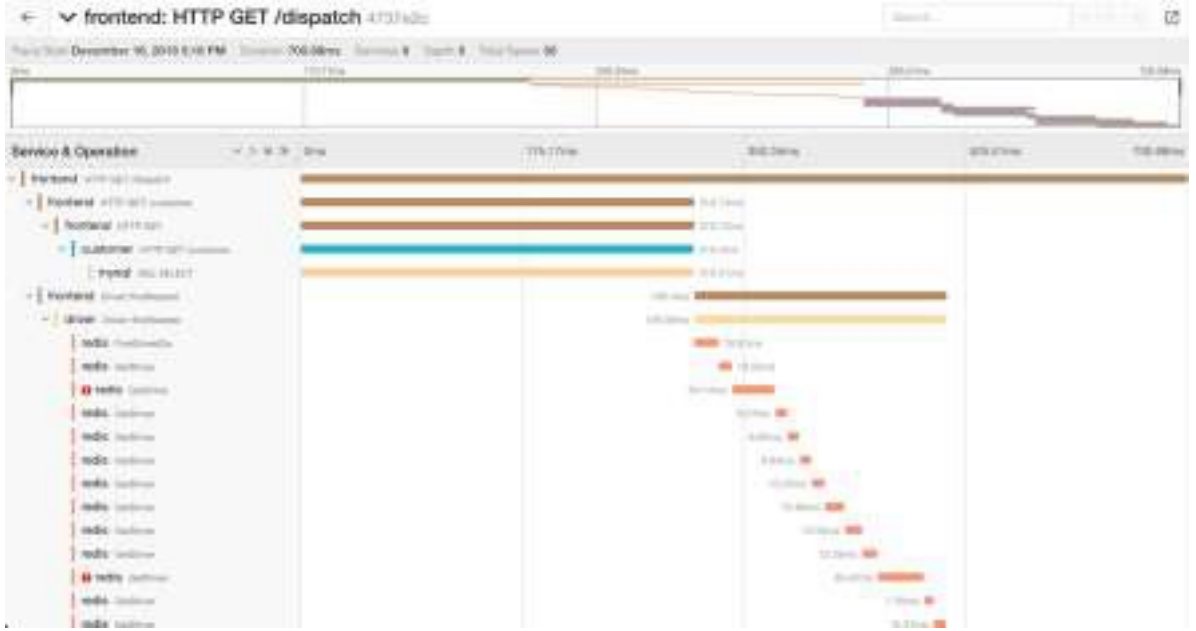
Different models driven by observability signals

RUM, Synthetics, NPM, APM, Infrastructure

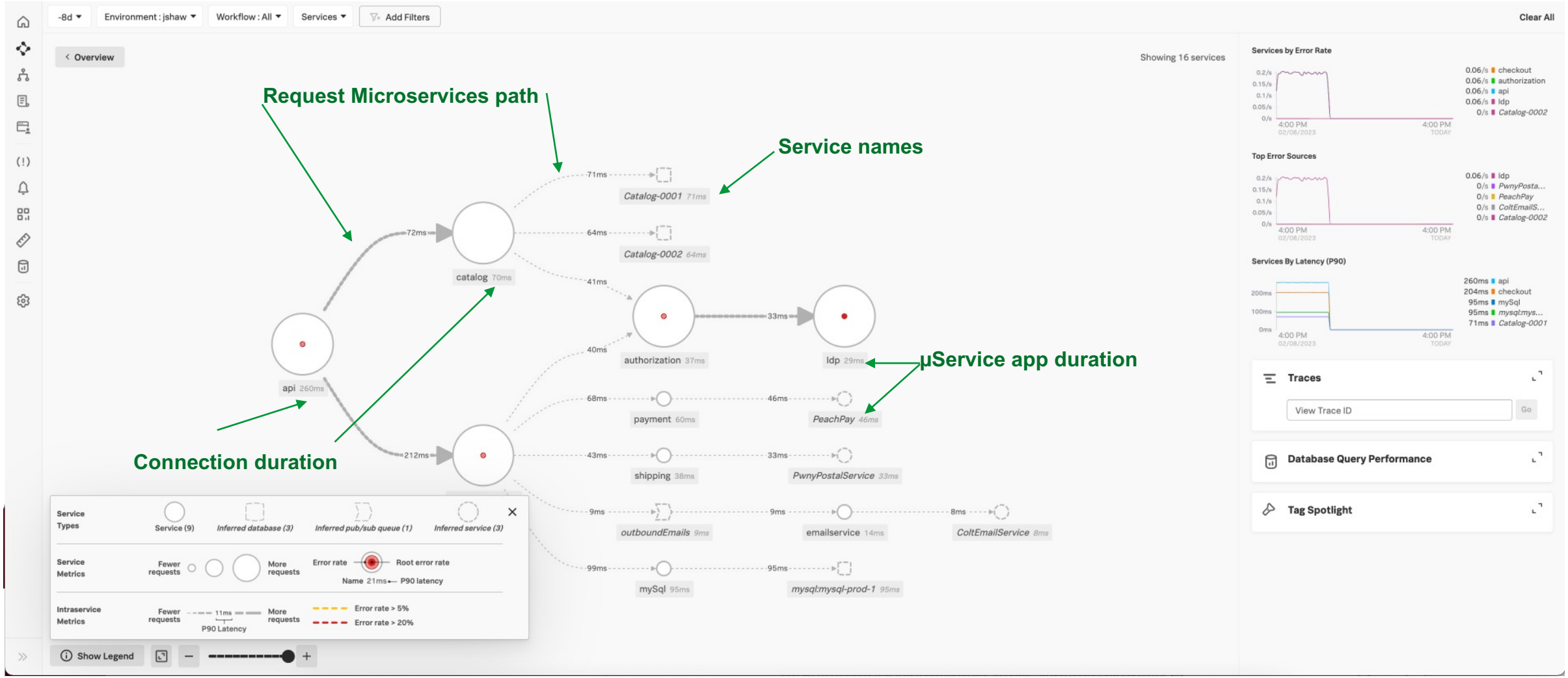


Tracing Concepts

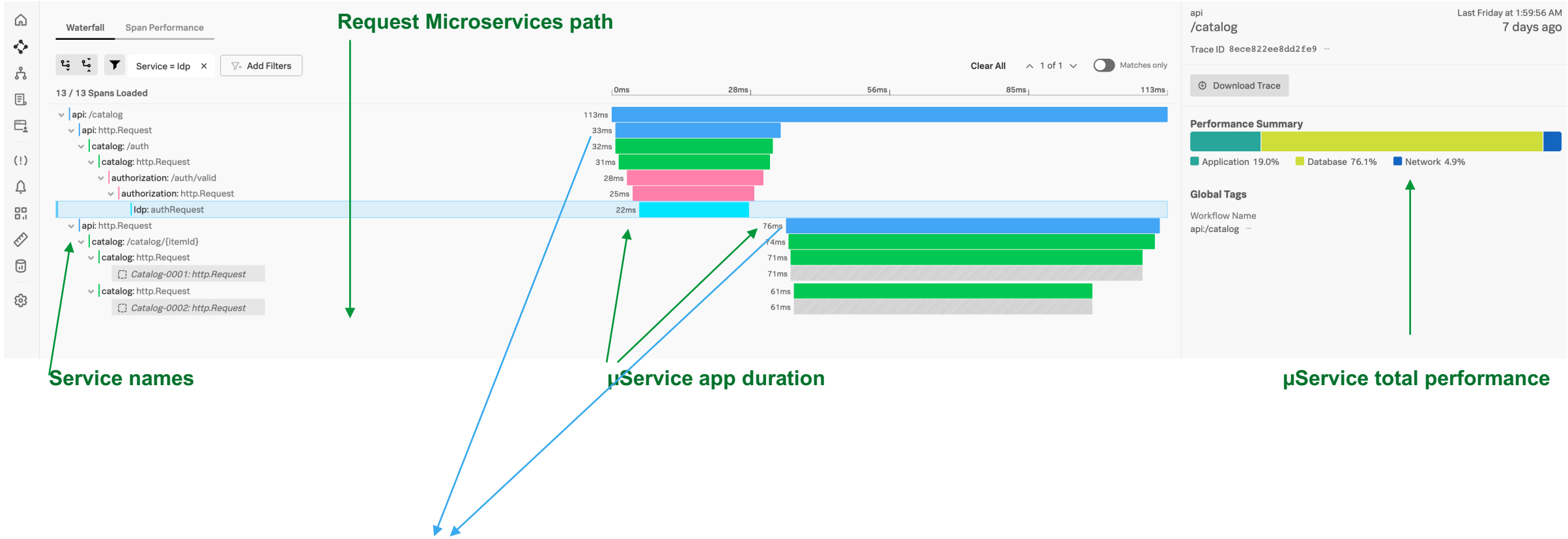
- Span
  - Represents a **single unit of work** in a system
- Trace
  - Defined implicitly by its spans
- Distributed Context
  - Tracing identifiers
  - Tags
  - Options that are propagated from parent to child spans



Let’s look at a trace

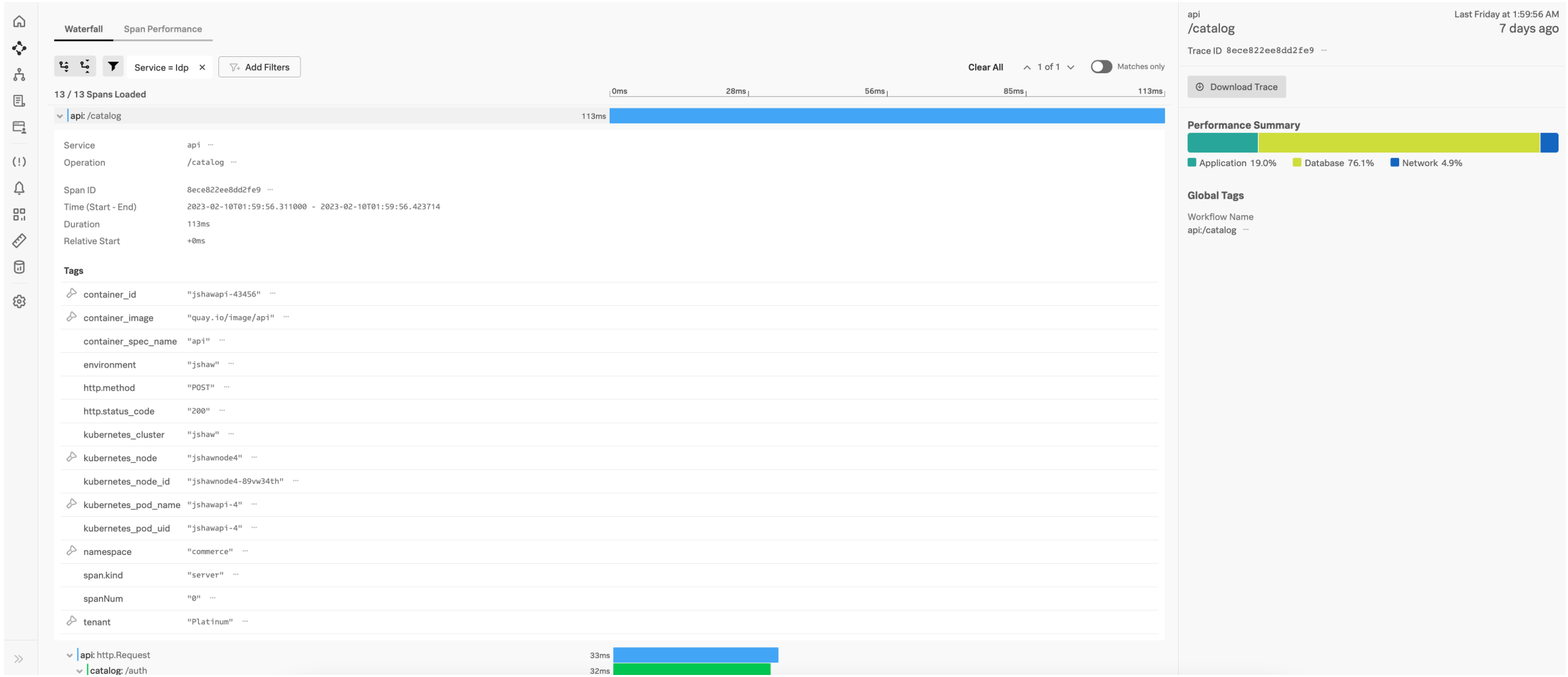


A different way of looking at a trace



Note the 2 spans makes up the trace duration (almost)

Observability includes baggage



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, async calls)

What this basically means

Traces

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

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

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

What problems are you trying to solve?

A tiny subset of answers*

- Performance issues
- Mean time to resolution and detection is too high
- Metrics and logs are missing valuable context
- More data types to provide better answers

A span for everything or bare minimum

It depends

Remember why you want to trace

Start with service boundaries and 3rd party calls

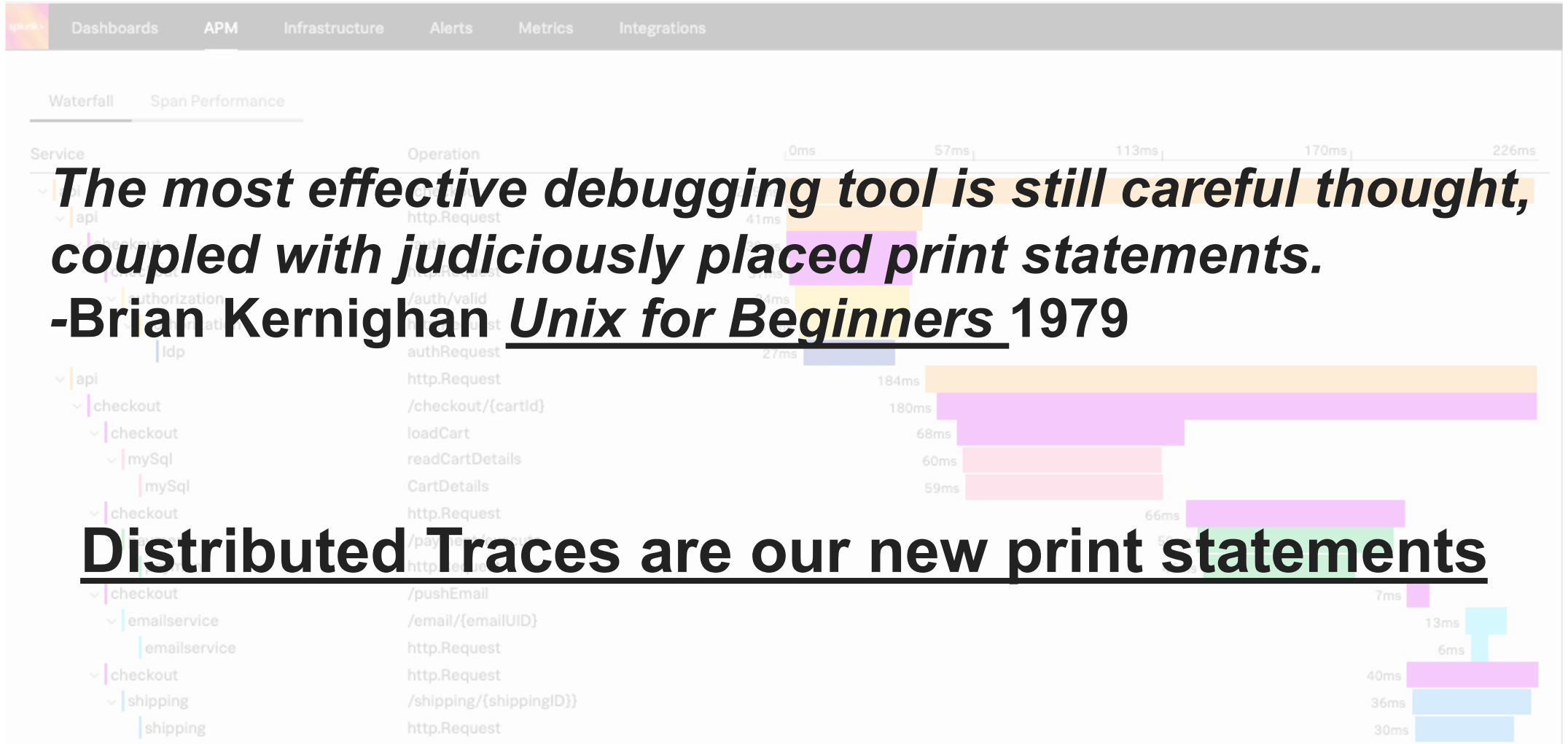
Iterative process

Beware of information overload

Distributed Tracing

- Gives insights into the app and its infrastructure
- Requires effort to return value
- Is a good ”User Happiness” proxy
- Does not magically solve your issues

Closing Thoughts



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

-Brian Kernighan *Unix for Beginners* 1979

Distributed Traces are our new print statements

THANKS!

https://www.linkedin.com/in/davemc