Observability, RED and Kubernetes

Adventures in Multi-Dimensional Observability

Dave McAllister, Technical Evangelist, Splunk

January 23rd, 2020

“You see, but you do not observe.”

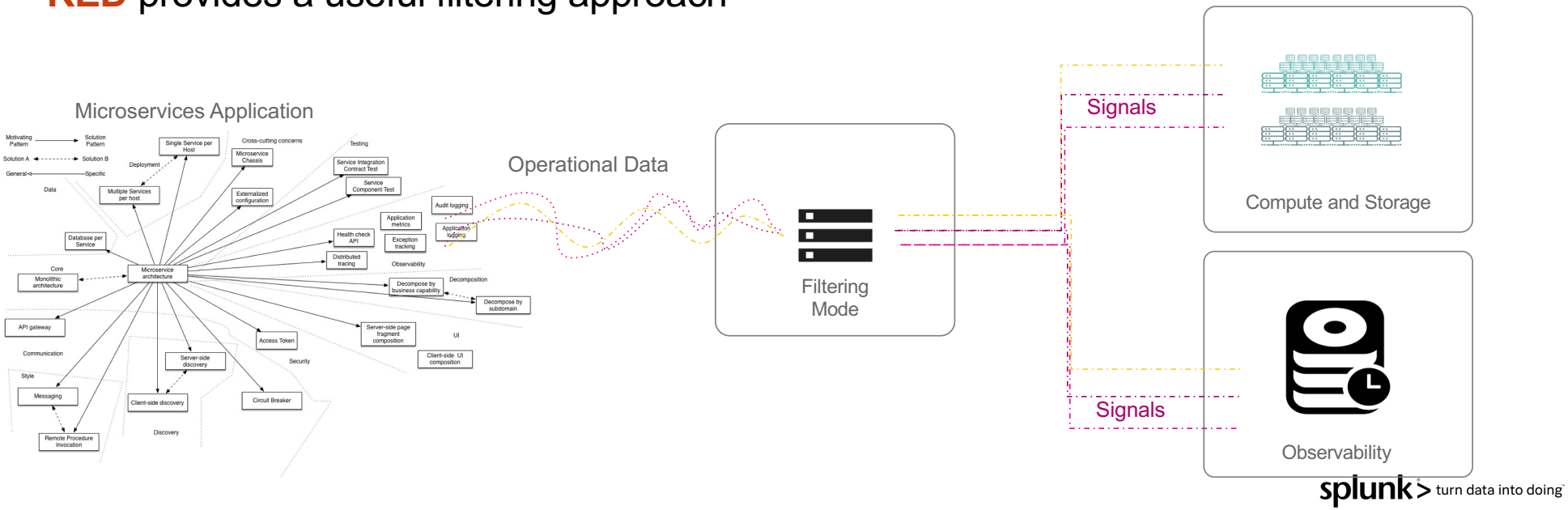
Sir Arthur Conan Doyle
A Scandal in Bohemia

splunk> turn data into doing

Observability is a Signal to Noise Problem

Actually, it is Signals not Noise

RED provides a useful filtering approach



A Brief View of Observability

TL;DR: Observability is a quality of software, services, platforms, or products that allows us to understand how systems are behaving.

For Engineering purposes: **Designing / defining the exposure of state variables in a manner to allow inference of internal behavior**

Observability is Not One Dimensional

- Recall “Internal States Inferred from External Outputs”
  - Observability is a property of the system—not a tool
- Should consist of logs, monitoring, events/tracing, and anything else
- Should include elements of metrics and time
- Should cross boundaries
  - Apps
  - Services
  - Disciplines
- Anything that slows you down is bad

Observability Isn't Just Monitoring

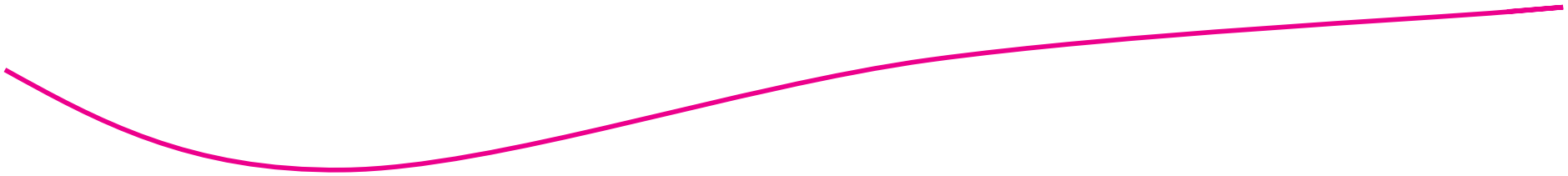
- A (system) attribute and a verb
- There aren't just three pillars, there are many signals
  - Metrics, logs, traces, events, errors, RPCs, core dumps, profiling data. From apps, services, containers, infrastructure…
- You need signals you haven't even thought of
- And they aren't static

Three Primary Pillars of Observability

Detect

Troubleshoot

Pinpoint



Metrics

Do I have a Problem

Traces

Where is the Problem

Logs

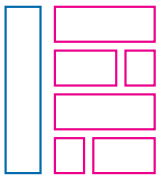
What is causing the problem?

Logs and Metrics and Traces, oh my!

- Everything is an event
- Events only exist if recorded/measured
  - Metrics are compact, efficient, but may not be sufficient (or complete)
  - Logs/events are full fidelity, but relatively bulky at full capture
  - Traces are cool, but require work (instrumentation) to establish

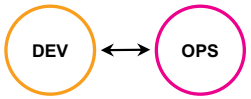
World of Operations & App Development

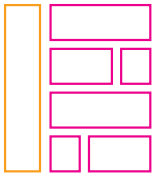
Retain & Optimize



Tightly Coupled Apps,
Slow Deployment Cycles

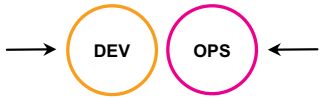
Lift & Shift



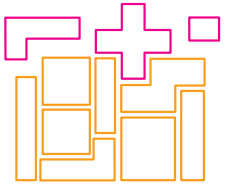


Primarily using
Cloud IaaS

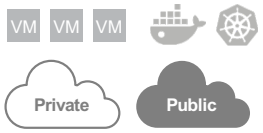
Re-Factor



Cloud Managed e.g. RDS,
DynamoDB, SaaS

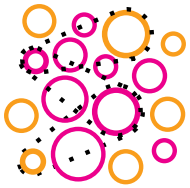


More Modular, but
Dependent App Components



Re-Architect / Cloud-Native

Cloud First Architecture



Loosely Coupled Microservices,
and Serverless Functions

splunk> turn data into doing

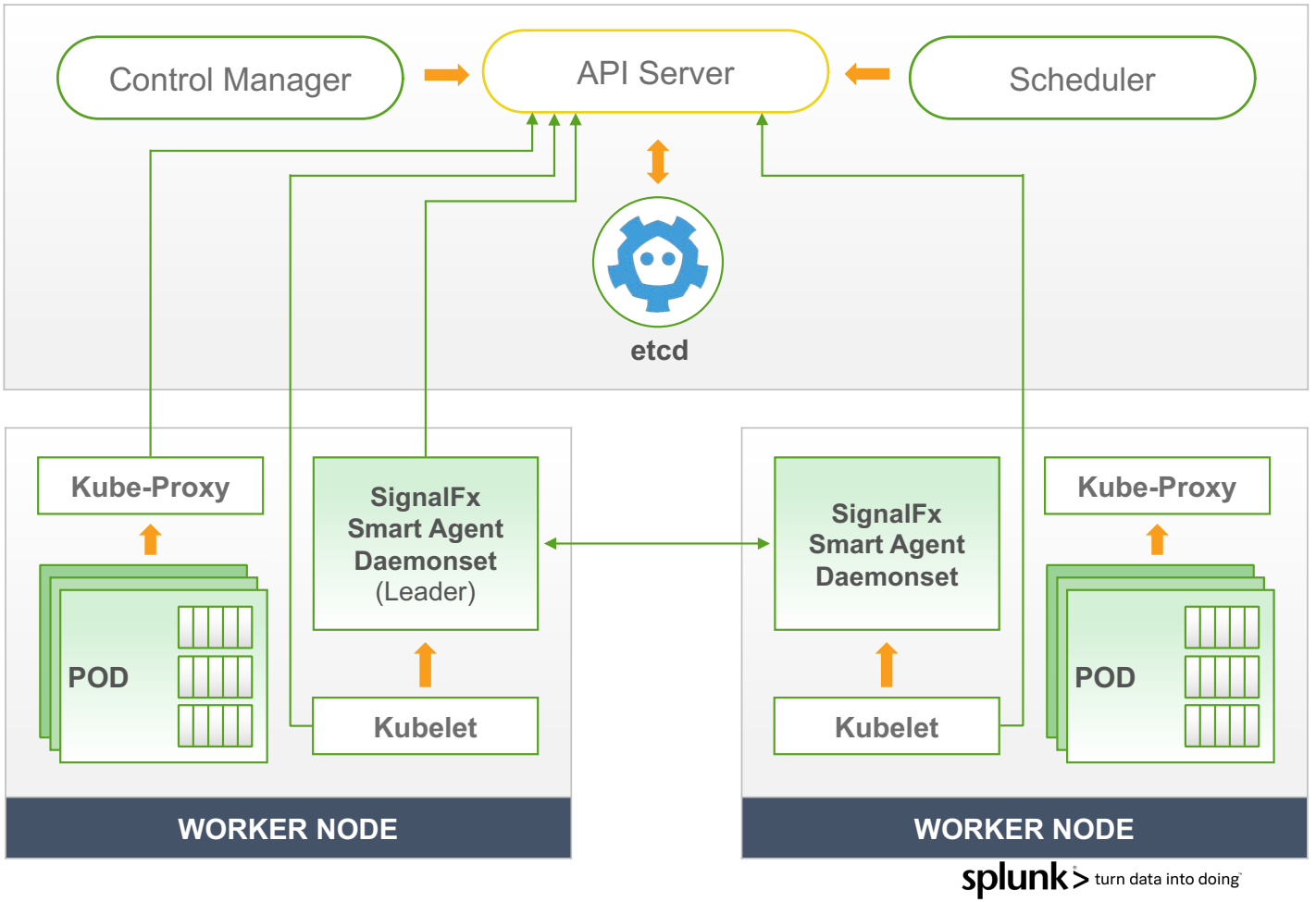
What a Microservices Architecture Looks Like

An architecture for building a continuously available **distributed system**
with a **distributed development** model

Sources: https://martinfowler.com/microservices/#what, https://articles.microservices.com/what-are-microservices-a-pragmatic-definition-1aa72839bc98

A Brief Look at Kubernetes

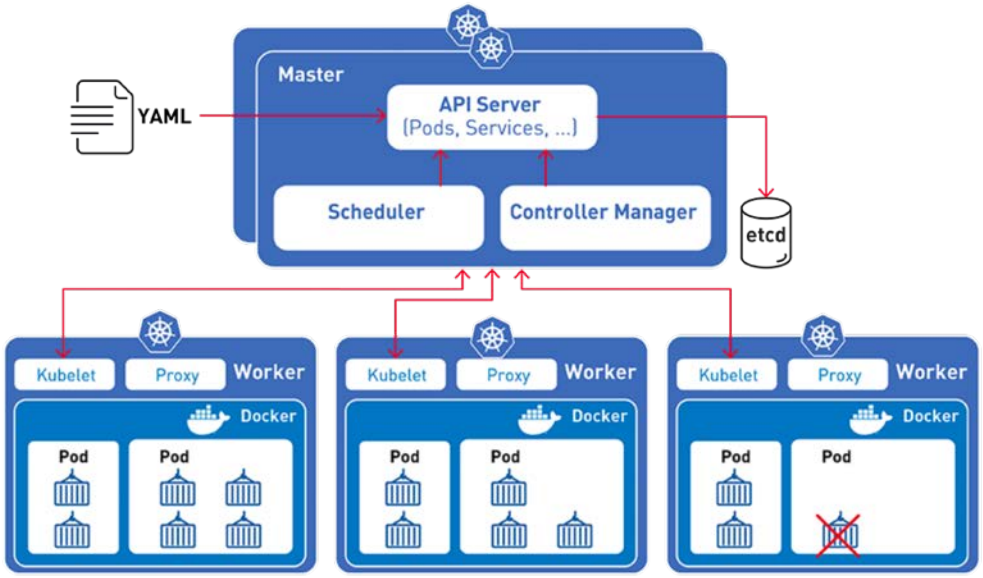
- Kubernetes orchestrates computing, networking, and storage infrastructure on behalf of user workloads
- You describe your cluster's desired state using Kubernetes API Objects
  - What workloads to run
  - What resources should it use
  - How many replicas to keep running
- Kubernetes Control Plane then works to make your Kubernetes cluster match the desired state
  - Starting/restarting containers
  - Scaling the number of replicas
  - Placing containers on the right node



K8s Increases Ephemerality and High-Churn

"Among all organizations using containers in production **68% use K8s**"
– HEPTIO THE STATE OF K8S REPORT 2018

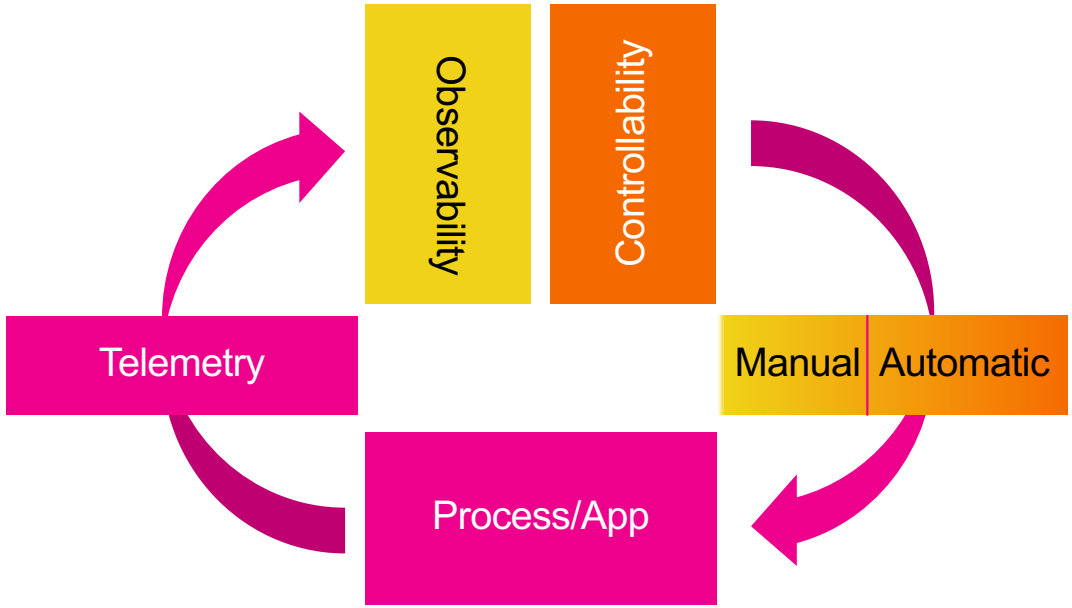
- Multiple layers of abstraction to monitor: containers, pods, clusters, nodes, namespaces, etc.
- Container spin-up and down in seconds
- Dynamic workload placement
- Challenges with monitoring end-to-end performance of distributed services



Observability and the Loop Dilemma

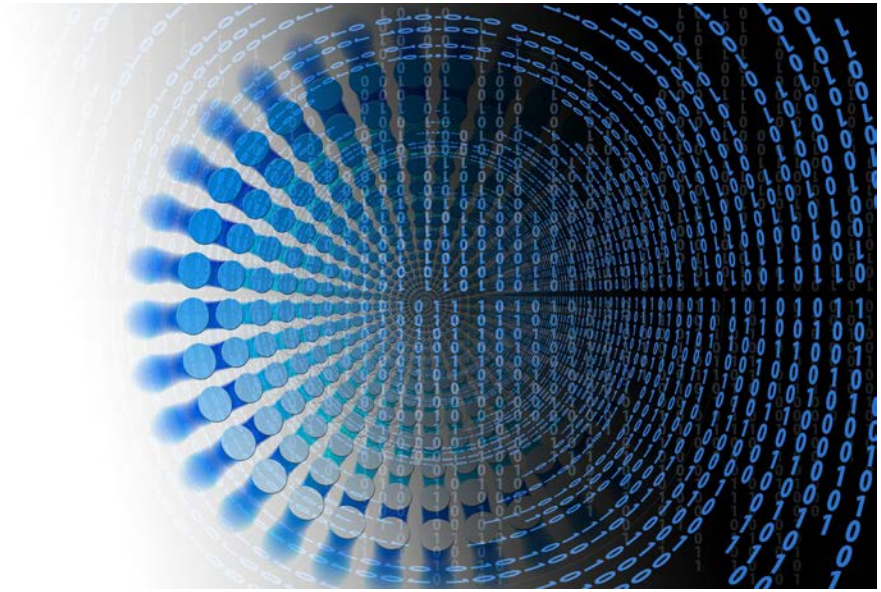
Even when it isn't a loop

- Duality of Observability is Controllability
- The Telemetry is our inputs
  - Logs
  - Traces
  - Metrics
  - And more
- Loops can be open or closed



Why RED and Kubernetes

- Complexity matters
  - Lots of moving items
  - Lots of interrelations
  - Lots of “Not there now”
- We need simplicity and abstraction to resolve clutter
- We need to retain complexity for “Gotchas” and “A-ha’s”



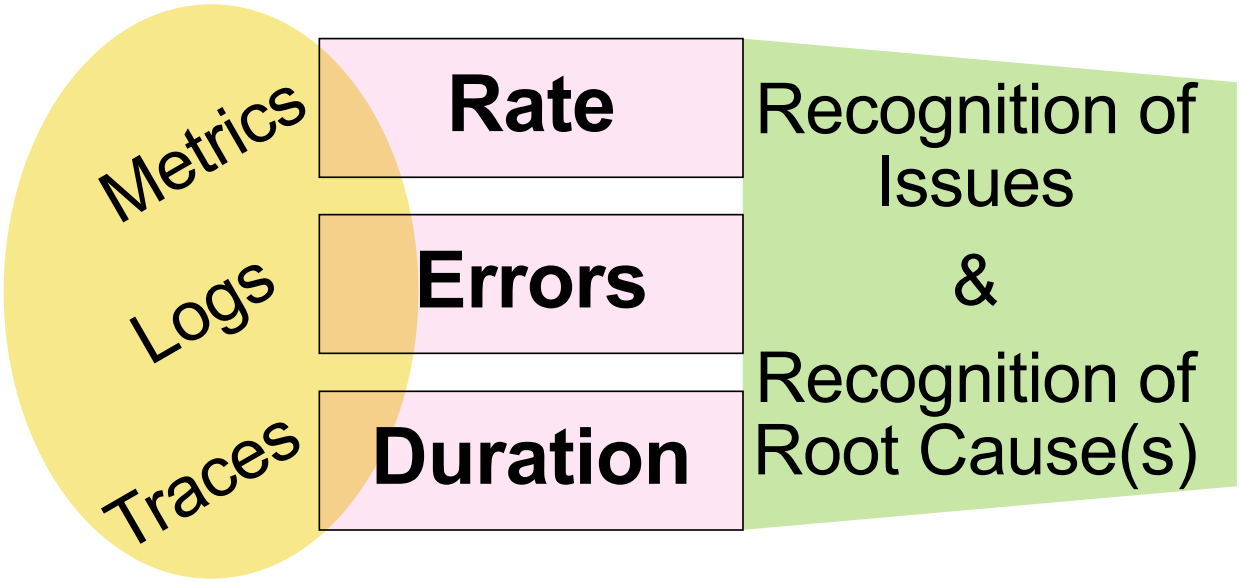
So what is RED?

- A subset of Google's golden signals (SRE-related)
  - Originally mentioned by WeaveWorks (H/T Tom Wilkie)
- Made up of rate, errors, duration
- Designed for request-driven systems, microservices

Service	Req/sec	Error Rate	P50 Duration	P90 Duration
api	9.9	51%	96ms	98ms
catalog	0.70	29%	74ms	75ms
checkout	9.3	8.5%	74ms	75ms
mangoDB	9.3	8.5%	32ms	50ms
payment	7.5	55%	50ms	51ms

RED & the Need for Multi-dimensional Capacity

Observability = Monitoring at the “Chuck Norris” level



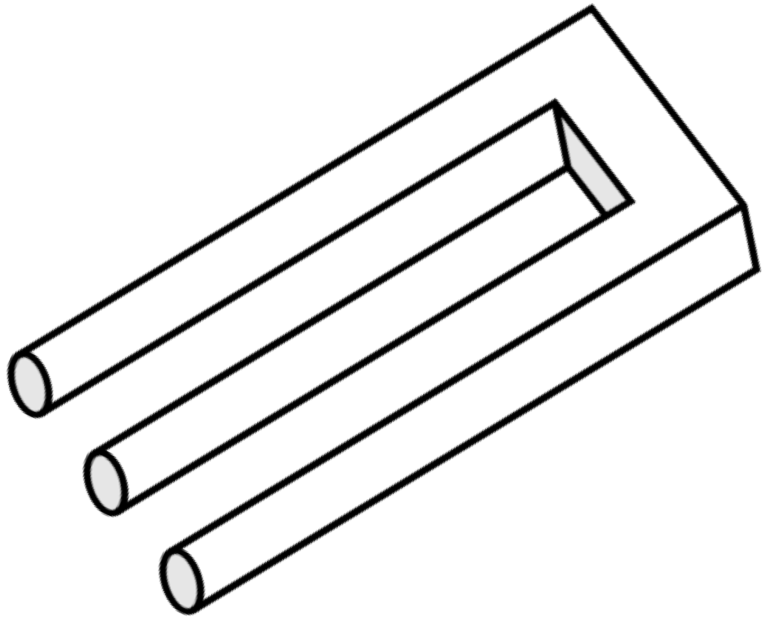
Rate

- Rate: number/size of requests on network and system
  - HTTP, SOAP, REST
  - Middleware messaging/queuing
  - API calls
  - Overhead of control structures like service meshes
- Any environment that can fail on peak traffic is a target for rate monitoring

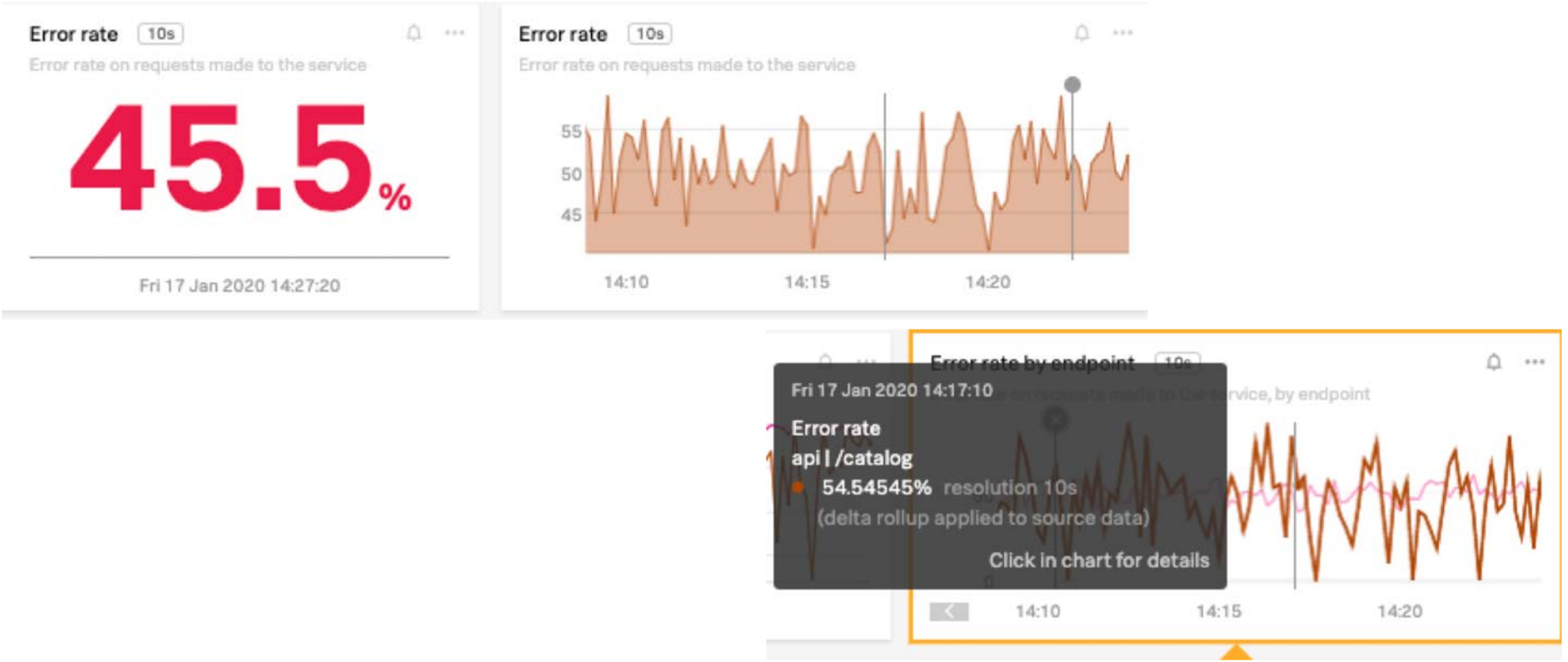
A Rate Example

Errors

- Errors: problems that cause an incorrect, incomplete or unexpected result
  - Code failures
  - Production load bugs
  - Peak load bugs
  - Communication woes
- Errors need:
  - Rapid Responses
  - Point Specific responses
- Need deep dive, high-fidelity
- Need ASAP



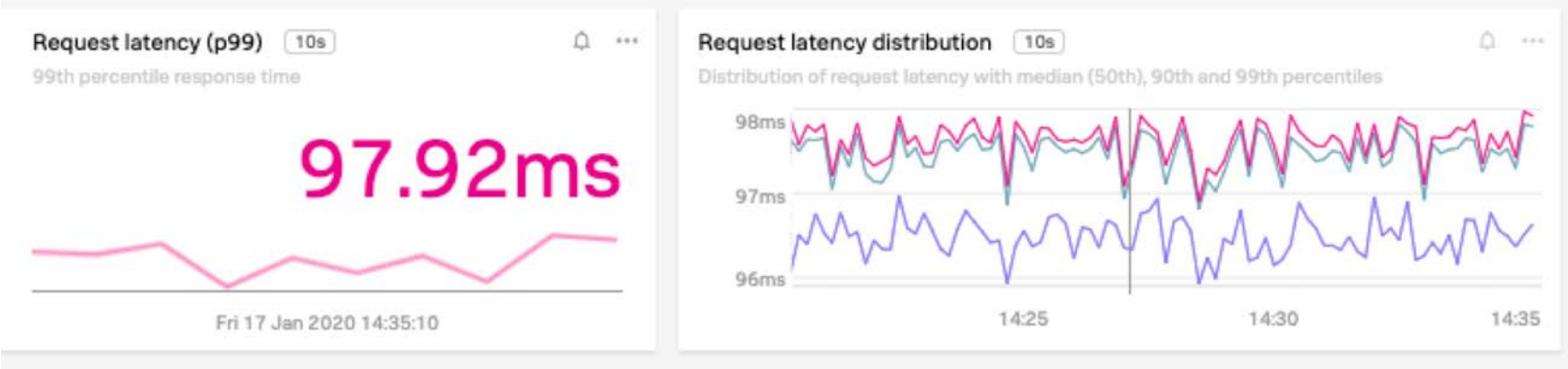
An Errors Example

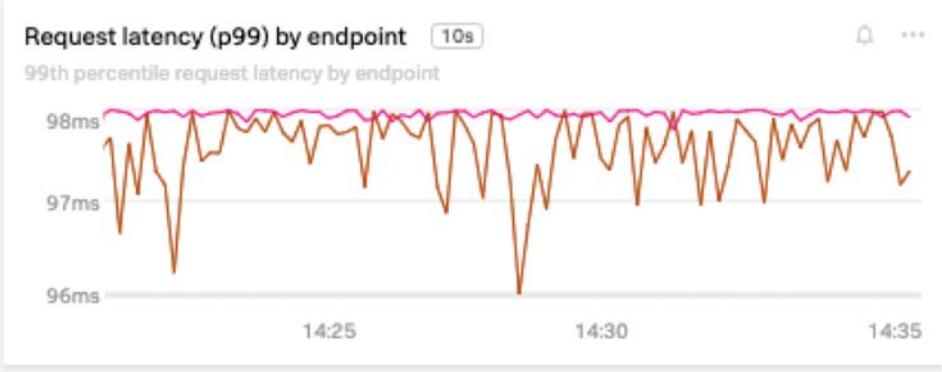


Duration

- It's all about time
- Both client-side and server-sides are important
  - But client side maybe more
- Usually (now) the domain and discussion of distributed request tracing
- **Bring events into causal order:**
  - When was the event? How long did it take?
  - How do I know it was slow?
  - Why did it take so long?
  - Which microservice was responsible?

Duration Example





Why RED?

- Easy to remember
- Reduces decision fatigue
- Drives standardization and consistency
- Helps with automation
- Serves as a proxy for user happiness

Crossing the Streams

It's in the last place you look

- Rate
  - Error or slow consumer
- Errors
  - Bandwidth limits or response time mismatch
- Duration
  - No response, limits from infrastructure

Challenges

- Traditional solutions have poor visualizations and are slow at scale
- Traditional solutions are hard to use
- Traditional solutions don’t allow easy disaggregation

So let’s take a look at how RED in monitoring can uncover a problem in microservices under Kubernetes

The Zen of Observability in Services

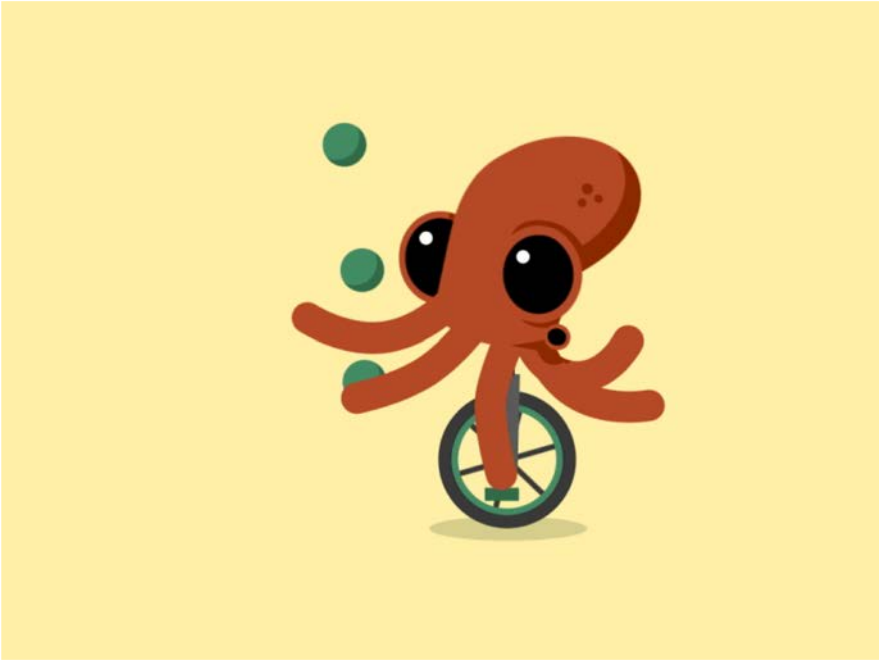
- Services observability has two perspectives on requests
- External (customer's) view is singular
  - Request, and its latency and success
- Operator's view is over a workload
  - Requests latency, rates, and concurrency
  - System resources/ components

Some Philosophy

- Instrumentation (by itself) is not an answer
  - It may help find an answer
- Metrics are powerful but not solely sufficient
  - Cardinality matters
- Observe the work, not the service
  - But observe the service anyway and particularly how the service responds to the workload
- Your goal is not “observability” but reducing
  - Mean time to detection
  - Mean time to response
  - Mean time to resolution

Summary

- Observability is more than monitoring
- RED wins for μservices-based apps
- RED simplifies the observability of Kubernetes while retaining insights
- Keep in mind that RED sections interact in interesting ways
- Find the right tool to give you clarity and insight



Thank You

Dave McAllister – davemc@splunk.com

splunk> turn data into doing™