Monitoring a production-ready microservices

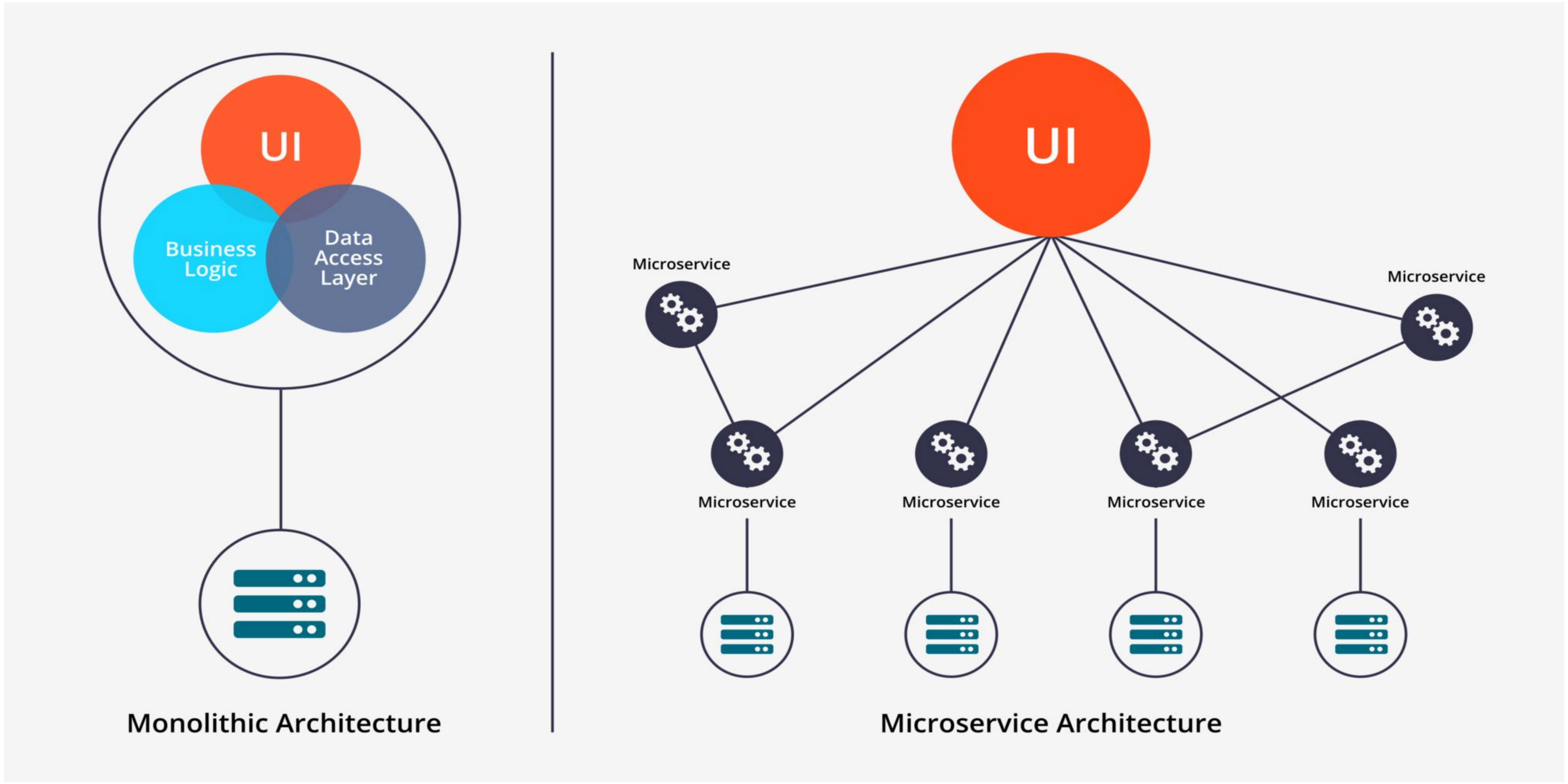
by Stanislav Kolenkin, Senior DevOps.

Introduction

Microservices are solving many architectural problems that have plagued developers and operations for years. But while the benefits are now clear, many professionals aren't fully aware of the many challenges that come with microservices. A significant portion of those center around monitoring and observability.

Microservices add complexity to your system, communication, and network, which makes monitoring more complicated.

Introduction



Introduction

You have a lot more systems

Introduction

50+ microservices

3 environments (DEV, STAGE, PROD)

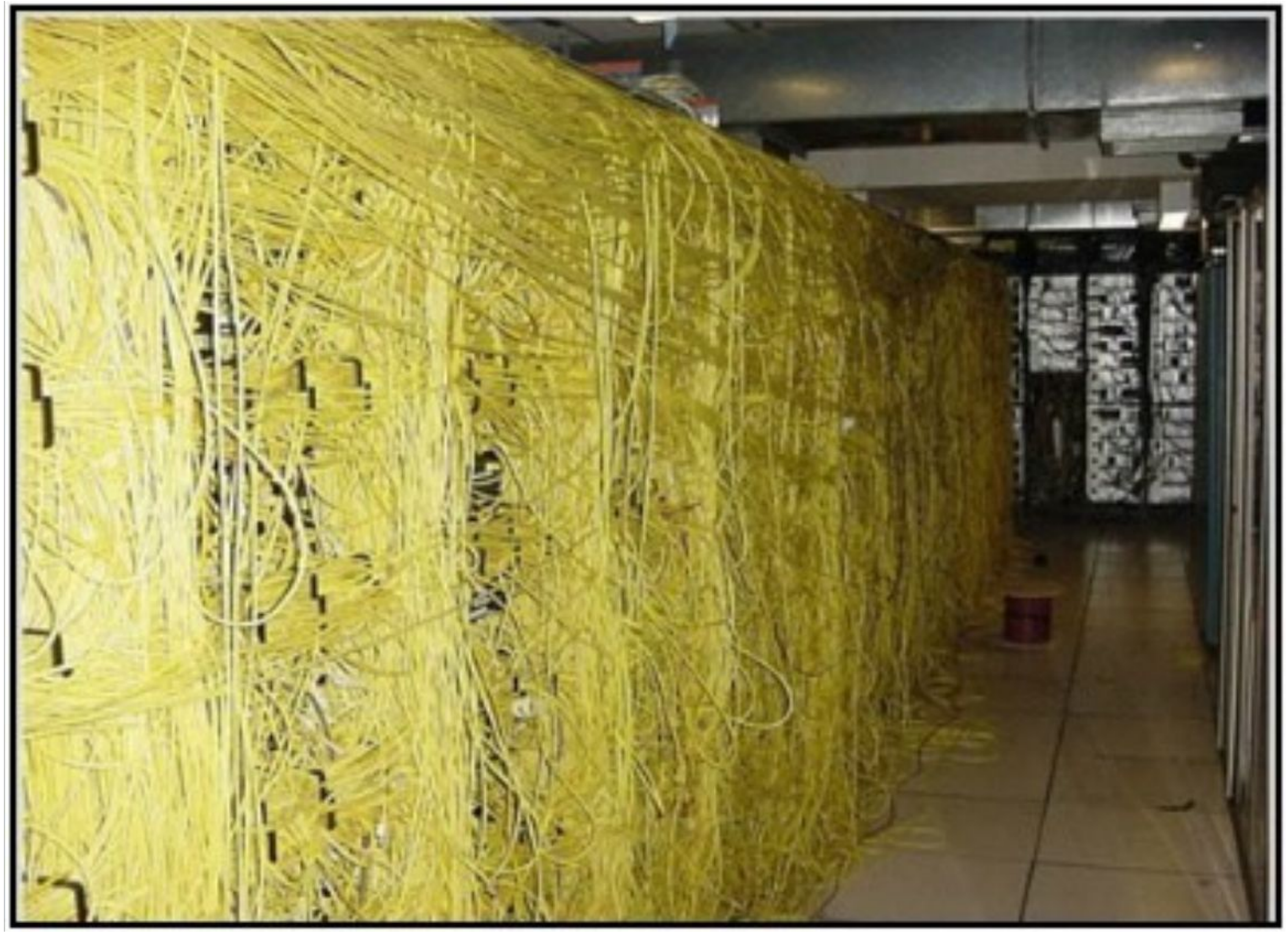
3 replica for each service

20+ check per service

...

Think about monitoring from the start

Monitoring



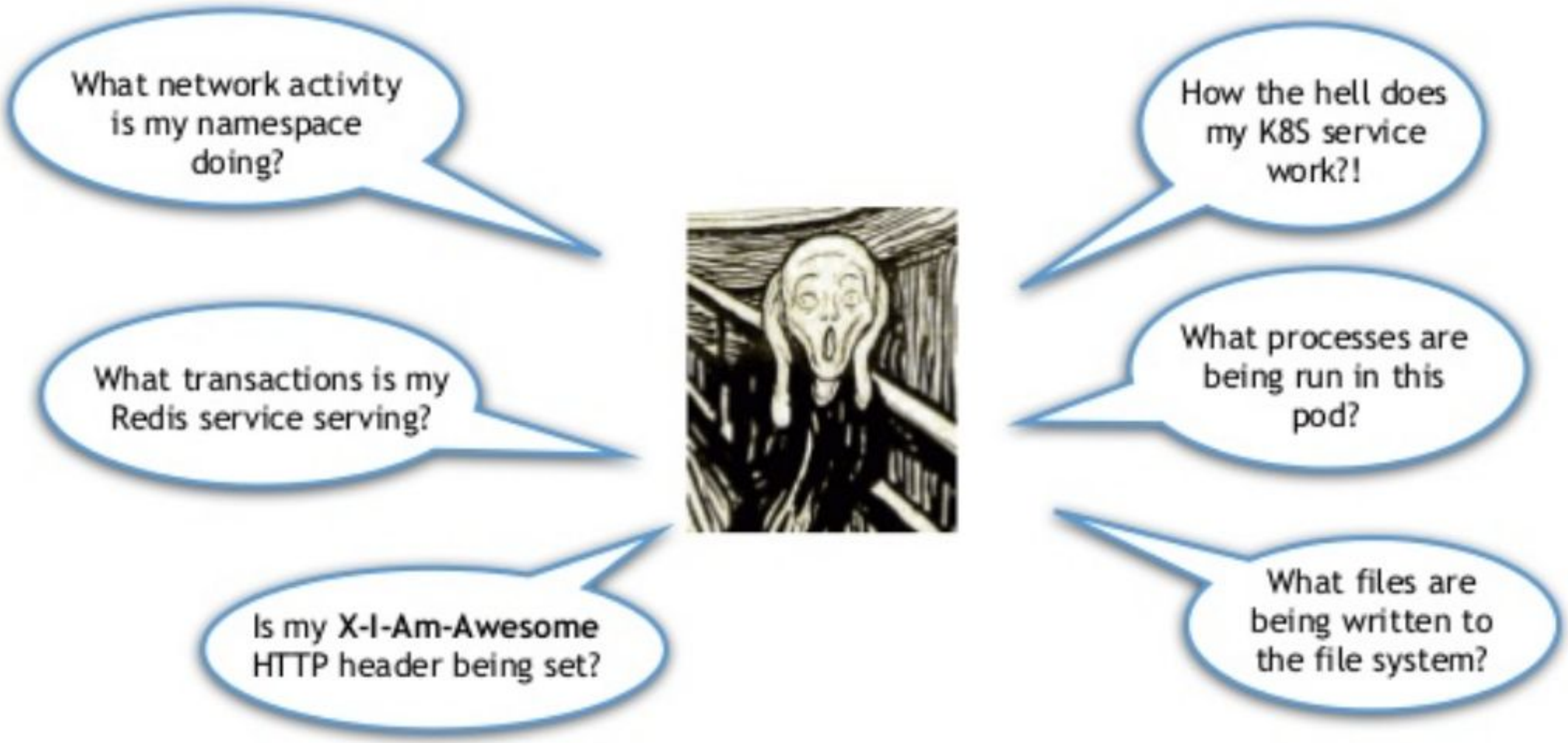
A network cable is unplugged

Monitoring

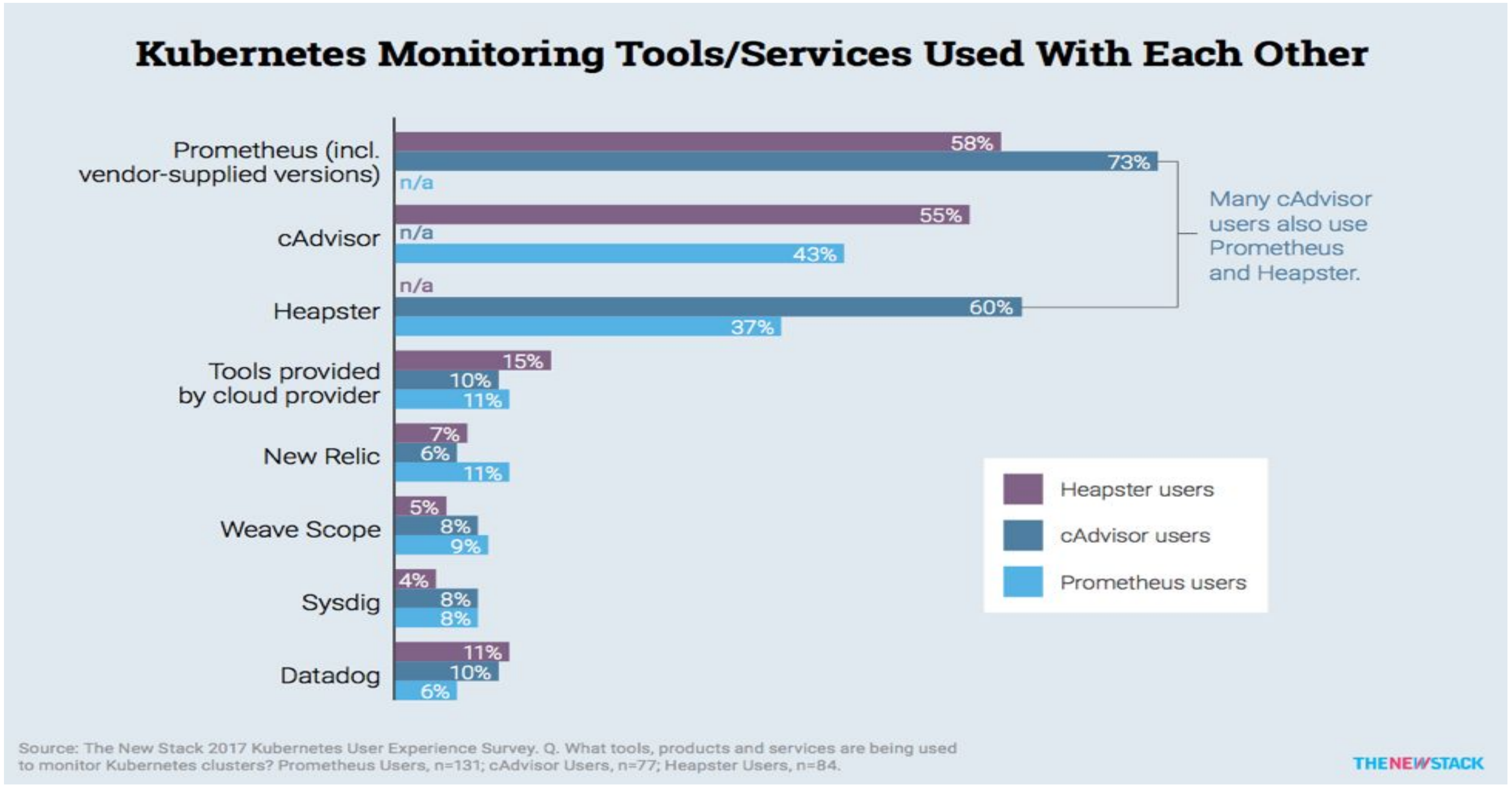
Kubernetes tools:

- Kubernetes UI
- cAdvisor
- Heapster
- Kubernetes dashboard
- Grafana
- ….

Monitoring

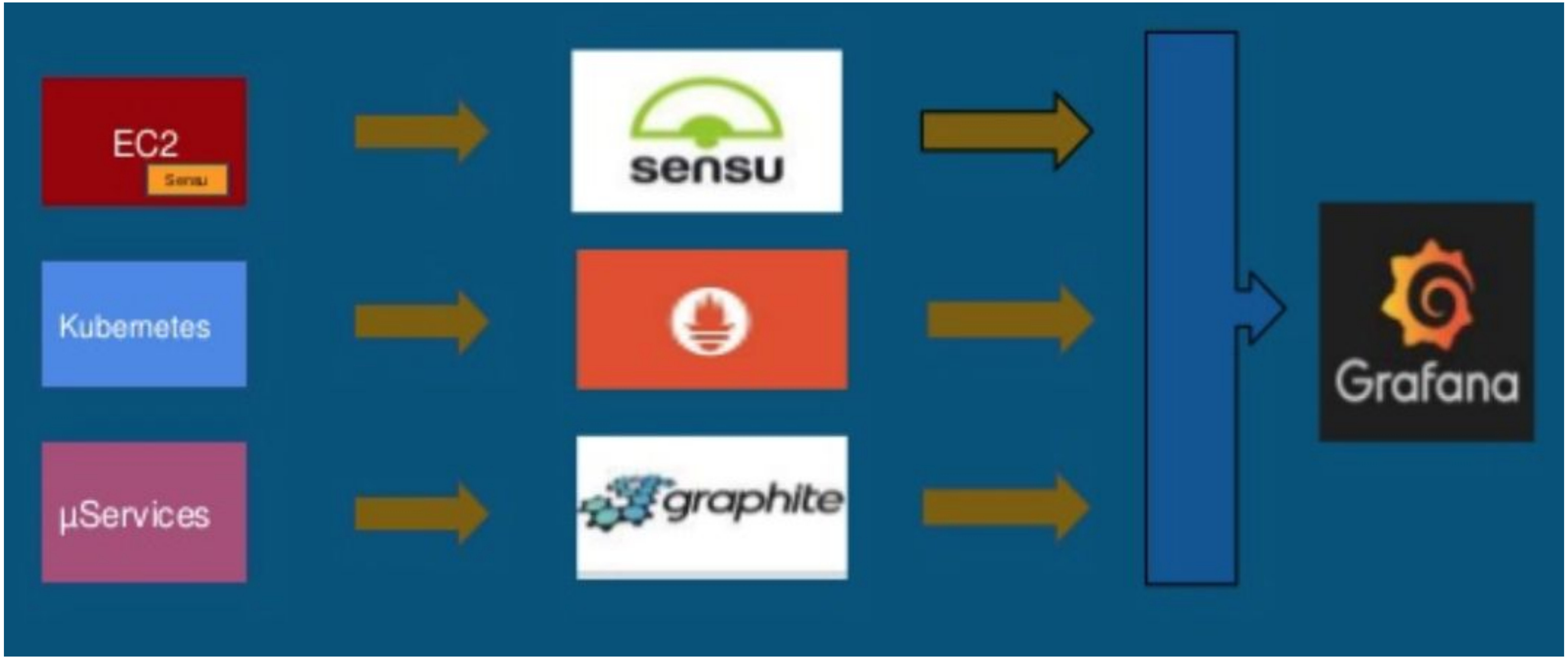

What network activity is my namespace doing?
How the hell does my K8S service work?!
What transactions is my Redis service serving?
What processes are being run in this pod?
Is my **X-I-Am-Awesome** HTTP header being set?
What files are being written to the file system?

Monitoring



Source: The New Stack 2017 Kubernetes User Experience Survey. Q. What tools, products and services are being used to monitor Kubernetes clusters? Prometheus Users, n=131; cAdvisor Users, n=77; Heapster Users, n=84.

Monitoring



Monitoring

Cons:

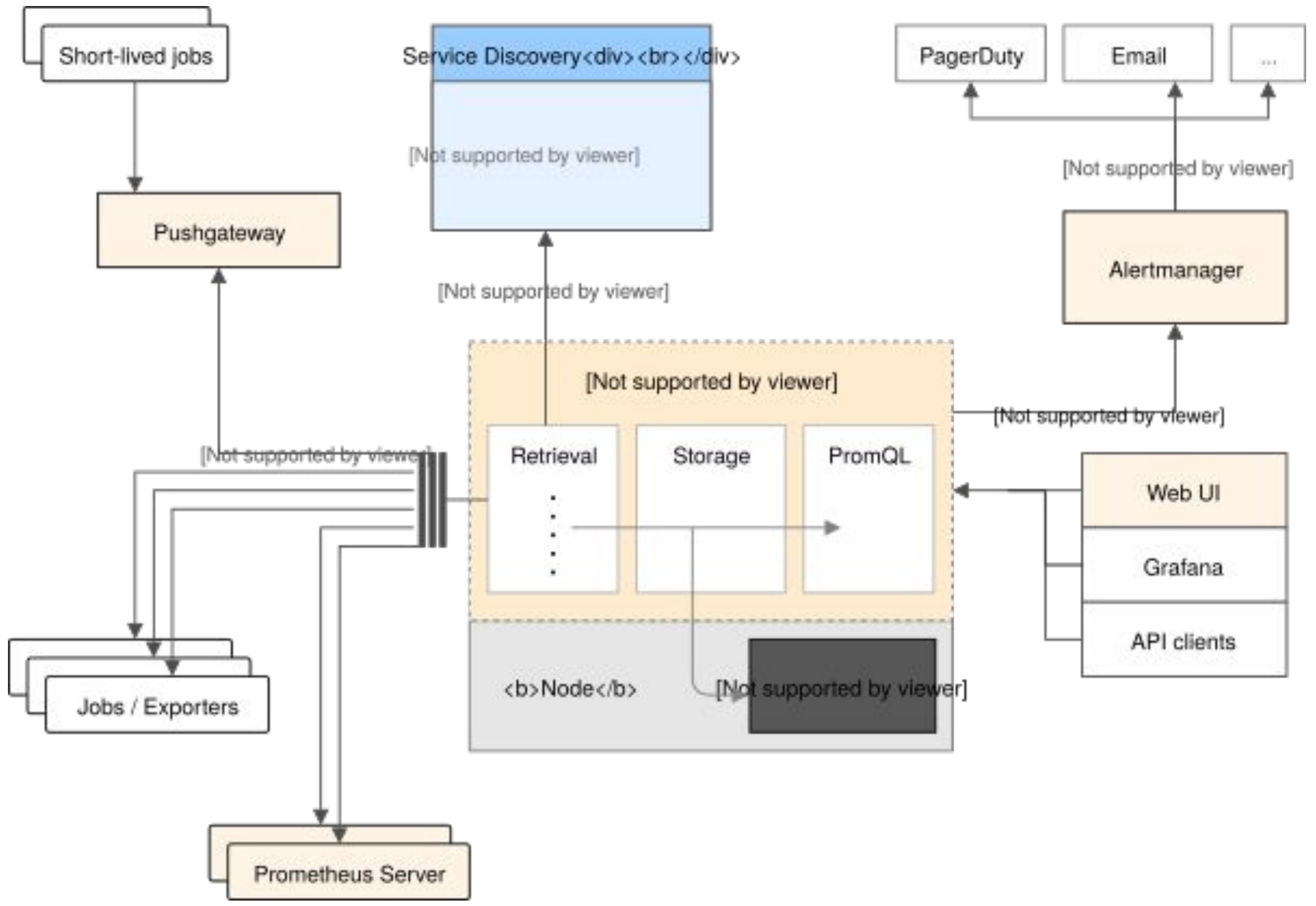
1. Multiple monitoring systems
2. Difficulty in troubleshooting
3. Additional Infrastructure cost to support three monitoring system
4. Graphite doesn’t provide pod level Application metrics
5. Infra team need to understand Sensu, Prometheus alerting
6. Application metrics are single dimension
7. Grafana alerting for Application metrics

Monitoring

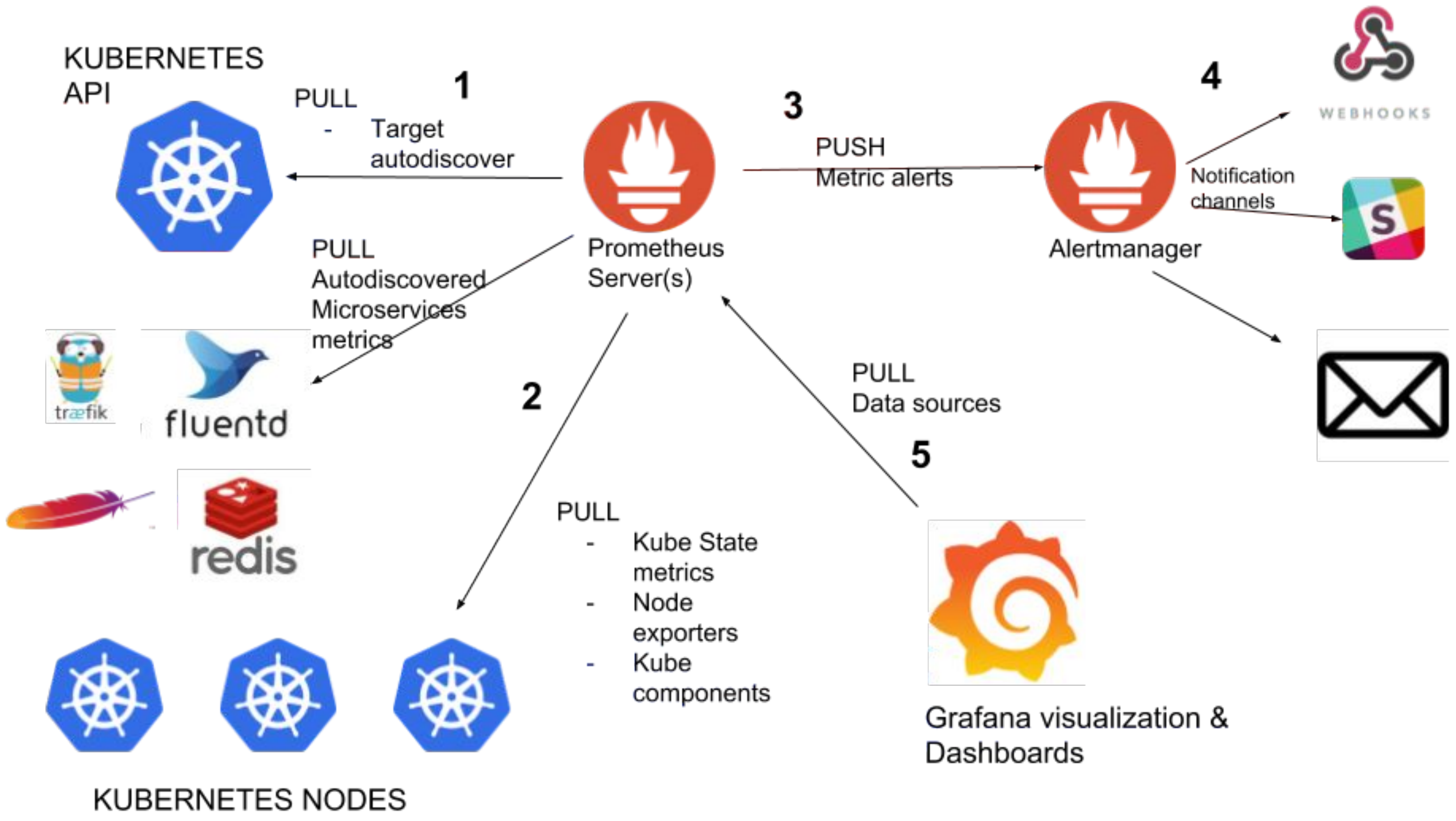
Prometheus:

- It developed as SoundCloud by ex-Googlers
- Prometheus is a close cousin of Kubernetes
- A multi-dimensional data model with time series data identified by metric name and key/value pairs
- Alerting and graphing are unified, using the same language
- Time series collection happens via a pull model over HTTP
- Targets are discovered via service discovery or static configuration
- Provides multiple exporters to send AWS EC2, Kafka, Redis, RMQ,... metrics
- You can export specific metrics from Prometheus to Kubernetes and use these metrics in HPA

Monitoring


Short-lived jobs
Pushgateway
Service Discovery<div>
</div>
[Not supported by viewer]
[Not supported by viewer]
PagerDuty
Email
...
[Not supported by viewer]
Alertmanager
[Not supported by viewer]
[Not supported by viewer]
[Not supported by viewer]
Retrieval
Storage
PromQL
Web UI
Grafana
API clients
Jobs / Exporters
<b>Node</b>
[Not supported by viewer]
Prometheus Server

Monitoring

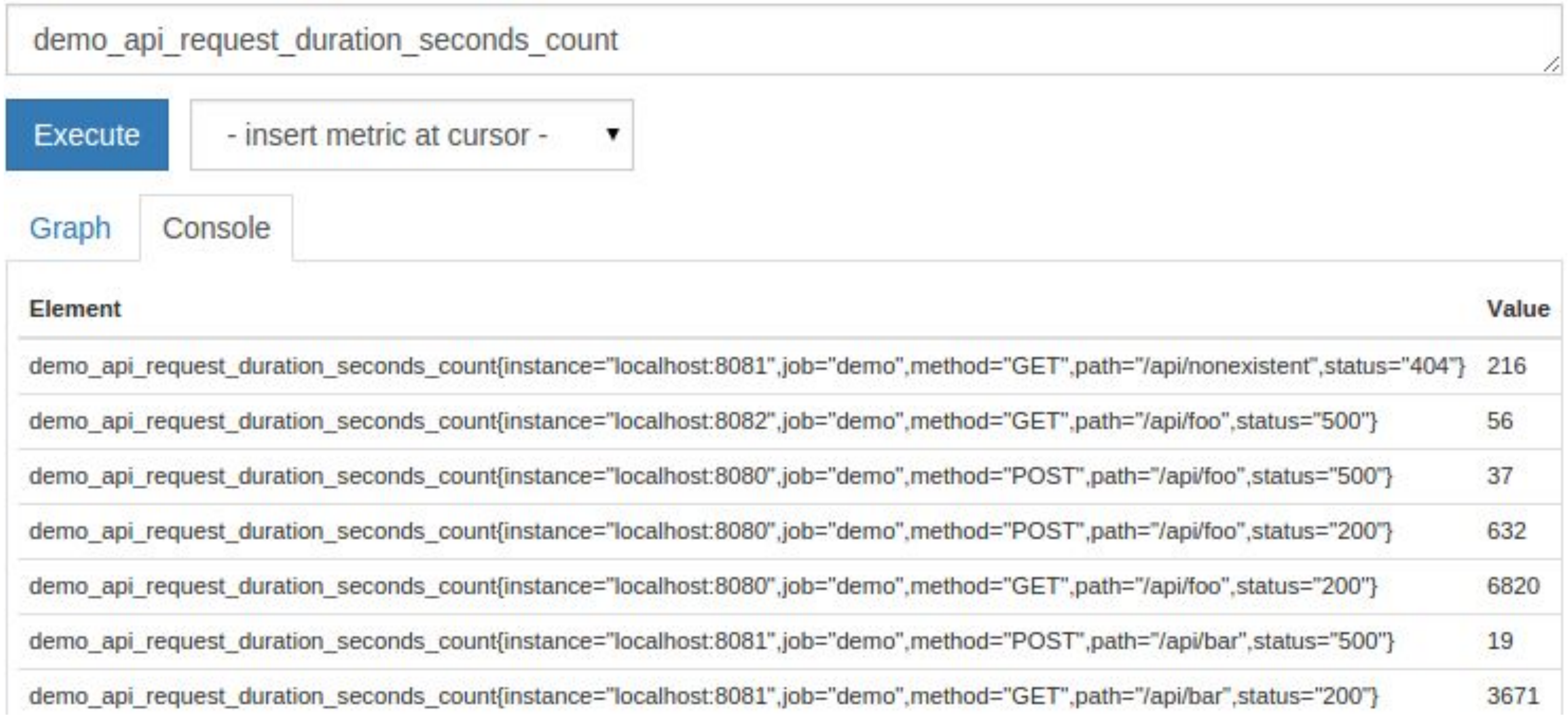

KUBERNETES API
1
PULL
- Target autodiscover
3
PUSH
Metric alerts
4
WEBHOOKS
Notification channels
Prometheus Server(s)
Alertmanager
PULL
Autodiscovered Microservices metrics
træfik
fluentd
redis
2
PULL
Data sources
5
PULL
- Kube State metrics
- Node exporters
- Kube components
Grafana visualization & Dashboards
KUBERNETES NODES

Monitoring

Metrics:

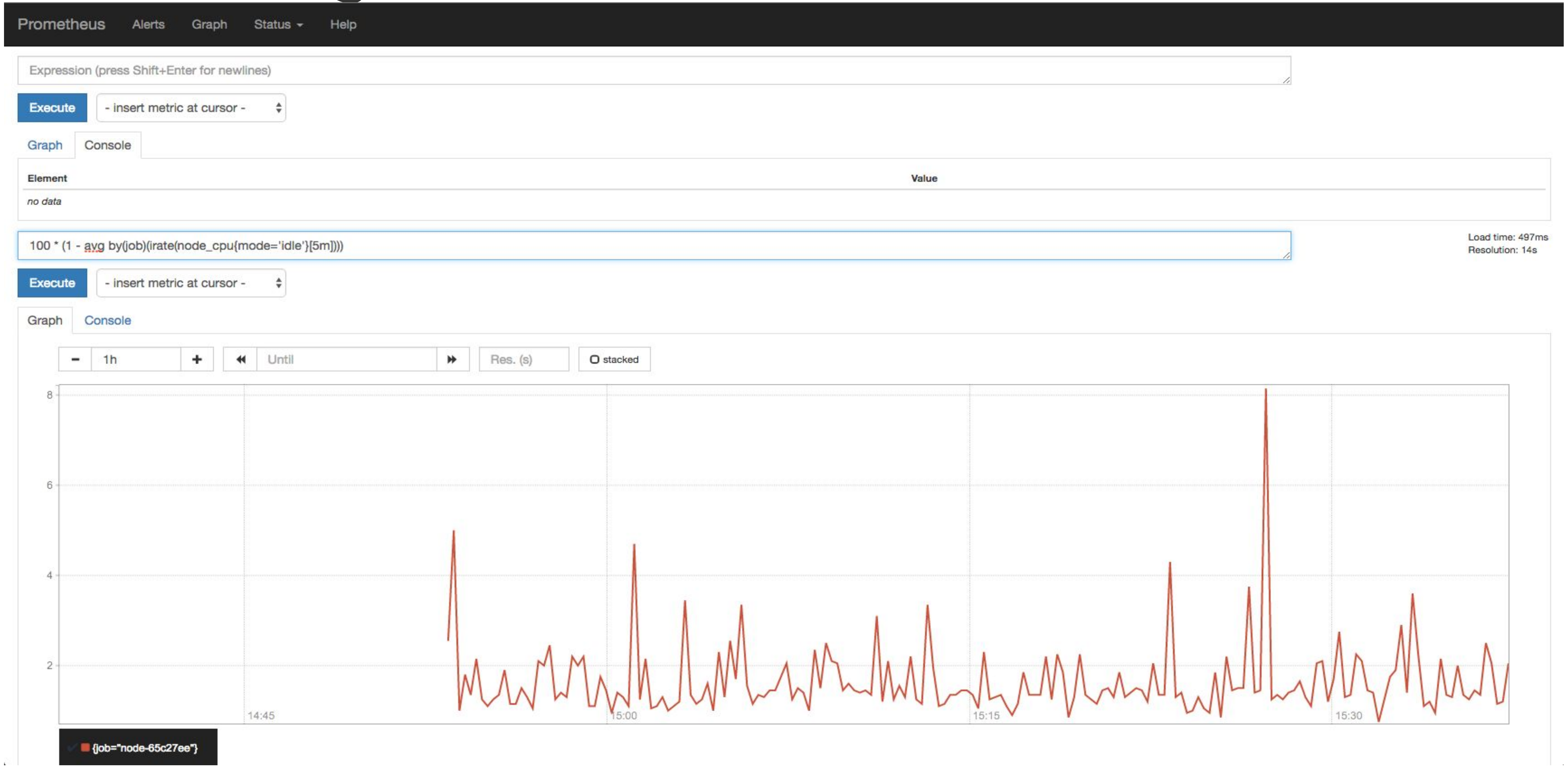
- CPU (system, user, nice, iowait, idle, irq, softirq, guest)
- Memory (Apps, Buffers, Cached, Free, SwapCached, Active, Inactive,...)
- Load
- Disk Space Used in percent
- Disk Utilization per Device
- Disk IOPS per device (read, write)
- Disk Throughput per Device (read, write)
- Context Switches
- Network Traffic (In, Out)
- UDP Stats
- Conttrack

Monitoring


demo_api_request_duration_seconds_count

Element	Value
demo_api_request_duration_seconds_count{instance="localhost:8081",job="demo",method="GET",path="/api/nonexistent",status="404"}	216
demo_api_request_duration_seconds_count{instance="localhost:8082",job="demo",method="GET",path="/api/foo",status="500"}	56
demo_api_request_duration_seconds_count{instance="localhost:8080",job="demo",method="POST",path="/api/foo",status="500"}	37
demo_api_request_duration_seconds_count{instance="localhost:8080",job="demo",method="POST",path="/api/foo",status="200"}	632
demo_api_request_duration_seconds_count{instance="localhost:8080",job="demo",method="GET",path="/api/foo",status="200"}	6820
demo_api_request_duration_seconds_count{instance="localhost:8081",job="demo",method="POST",path="/api/bar",status="500"}	19
demo_api_request_duration_seconds_count{instance="localhost:8081",job="demo",method="GET",path="/api/bar",status="200"}	3671

Monitoring


Prometheus
Alerts
Graph
Status
Help
Expression (press Shift+Enter for newlines)
Execute
- insert metric at cursor -
Graph
Console

Element	Value
no data	

```
100 * (1 - avg by(job)(irate(node_cpu{mode='idle'}[5m])))
```

Load time: 497ms
Resolution: 14s

Monitoring

Recap:

- Monitoring is essential to run, understand and operate services.
- Prometheus
  - Client instrumentation
  - Scrape configuration
  - Querying
  - Dashboards
  - Alert rules
- Important Metrics
  - Four golden signals: Latency, Traffic, Error, Saturation
- Best Practices

Logging

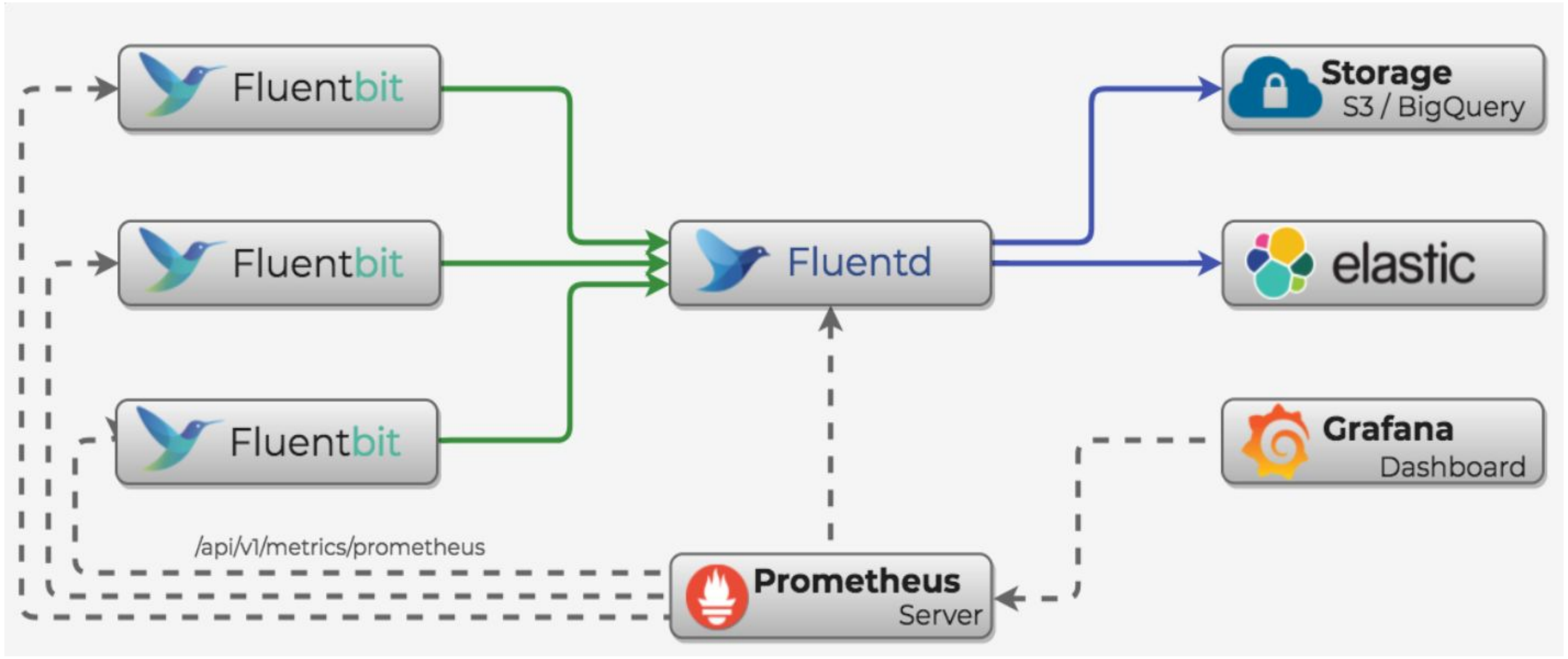
Your pods logs exist on the node only as long as the pod itself exists on the Kubernetes node; this means, when there’s an eviction event — for example, rescheduling pods from a tainted node-, and the pods are drained, the logs are taken with it. Persisting this data, however, can be useful, and doing this requires a little bit of iteration on the logging architecture. Persistence aside, this also allows you to leverage more advanced tooling around your log data once aggregated.

How your cluster ships logs to a logging backend will vary in small ways, but typically will come down to the following process; a DaemonSet (meaning a single pod that will run on as many nodes that exist in the cluster; if you have 4 nodes, you have 4 instances, and if you remove a node, that 4th instance is not rescheduled) is used to run a logging agent, and that agent will ship logs to your logging backend.

Logging

Unfortunately, Kubernetes doesn’t provide many configuration options that pertain to logging. Docker offers multiple logging drivers, but we can’t configure them via Kubernetes. Thankfully, there are already several open source solutions. When it comes to logging, our favorite tools are **Fluentd, Fluent-bit and Filebeat**.

Logging


Fluentbit
Fluentbit
Fluentbit
Fluentd
Storage
S3 / BigQuery
elastic
Grafana
Dashboard
/api/v1/metrics/prometheus
Prometheus
Server

Logging: Fluentd

Fluentd is an open source log collector, processor, and aggregator that was created back in 2011 by the folks at Treasure Data. Written in Ruby, Fluentd was created to act as a unified logging layer — a one-stop component that can aggregate data from multiple sources, unify the differently formatted data into JSON objects and route it to different output destinations.

Design wise — performance, scalability, and reliability are some of Fluentd's outstanding features. A vanilla Fluentd deployment will run on ~40MB of memory and is capable of processing above 10,000 events per second. Adding new inputs or outputs is relatively simple and has little effect on performance. Fluentd uses disk or memory for buffering and queuing to handle transmission failures or data overload and supports multiple configuration options to ensure a more resilient data pipeline.

Logging: Logstash

Logstash is not the oldest shipper of this list (that would be syslog-ng, ironically the only one with “new” in its name), but it’s certainly **the best known**. That’s because **it has lots of plugins**: inputs, codecs, filters and outputs. Basically, you can take pretty much any kind of data, enrich it as you wish, then push it to lots of destinations.

Logstash Advantages:

Logstash’s main strongpoint is **flexibility, due to the number of plugins**.

Logstash Disadvantages:

Logstash’s biggest con or “Achille’s heel” has always been **performance and resource consumption** (the default heap size is 1GB).

Logging: Fluentd

Fluentd has been around for some time now and has developed a rich ecosystem consisting of more than 700 different plugins that extend its functionality. Fluentd is the de-facto standard log aggregator used for logging in Kubernetes and as mentioned above, is one of the widely used Docker images.

Logging: Fluent Bit

Fluent Bit is an open source log collector and processor also created by the folks at Treasure Data in 2015. Written in C, Fluent Bit was created with a specific use case in mind — highly distributed environments where limited capacity and reduced overhead (memory and CPU) are a huge consideration.

To serve this purpose, Fluent Bit was designed for high performance and comes with a super light footprint, running on ~450KB only. An abstracted I/O handler allows asynchronous and event-driven read/write operations. For resiliency and reliability, various configuration option are available for defining retries and the buffer limit.

Logging: Fluent Bit

Fluent Bit is also extensible, but has a smaller eco-system compared to Fluentd. Inputs include syslog, tcp, systemd/journald but also CPU, memory, and disk. Outputs include Elasticsearch, InfluxDB, file and http. For Kubernetes deployments, a dedicated filter plugin will add metadata to log data, such as the pod’s name and namespace, and the containers name/ID.

Logging: Filebeat

Filebeat is a lightweight logshipper for logstash. It can be installed as agents on your servers to collect operational data. Since it is lightweight it does not consume system resources compared to Logstash. It has a number of modules that can help in parsing, collecting and visualization of logs. Moreover, it is container ready so in case if you are looking to build a container based solution you can run Filebeat inside a container in the same Virtual Machine

Logging: Compare

	Fluentd	Fluent Bit	Filebeat
Scope	Containers/Servers	Containers/Servers	Containers/Servers
Language	C&Ruby	C	GO
memory on start	~40MB	~450KB	~50-200MB
Performance	High Performance	High Performance	High Performance
Dependencies	Built as a Ruby Gem, It requires a certain number of gems	Zero dependencies, unless some special plugin requires them	Zero dependencies
Plugins	More than 650 plugins available	Around 35 plugins available	17 internal modules
License	Apache Licence 2.0	Apache Licence 2.0	Apache Licence 2.0

Logging: Conclusion

Logstash, fluentd, Fluent Bit, Filebeat have their own features as follows:

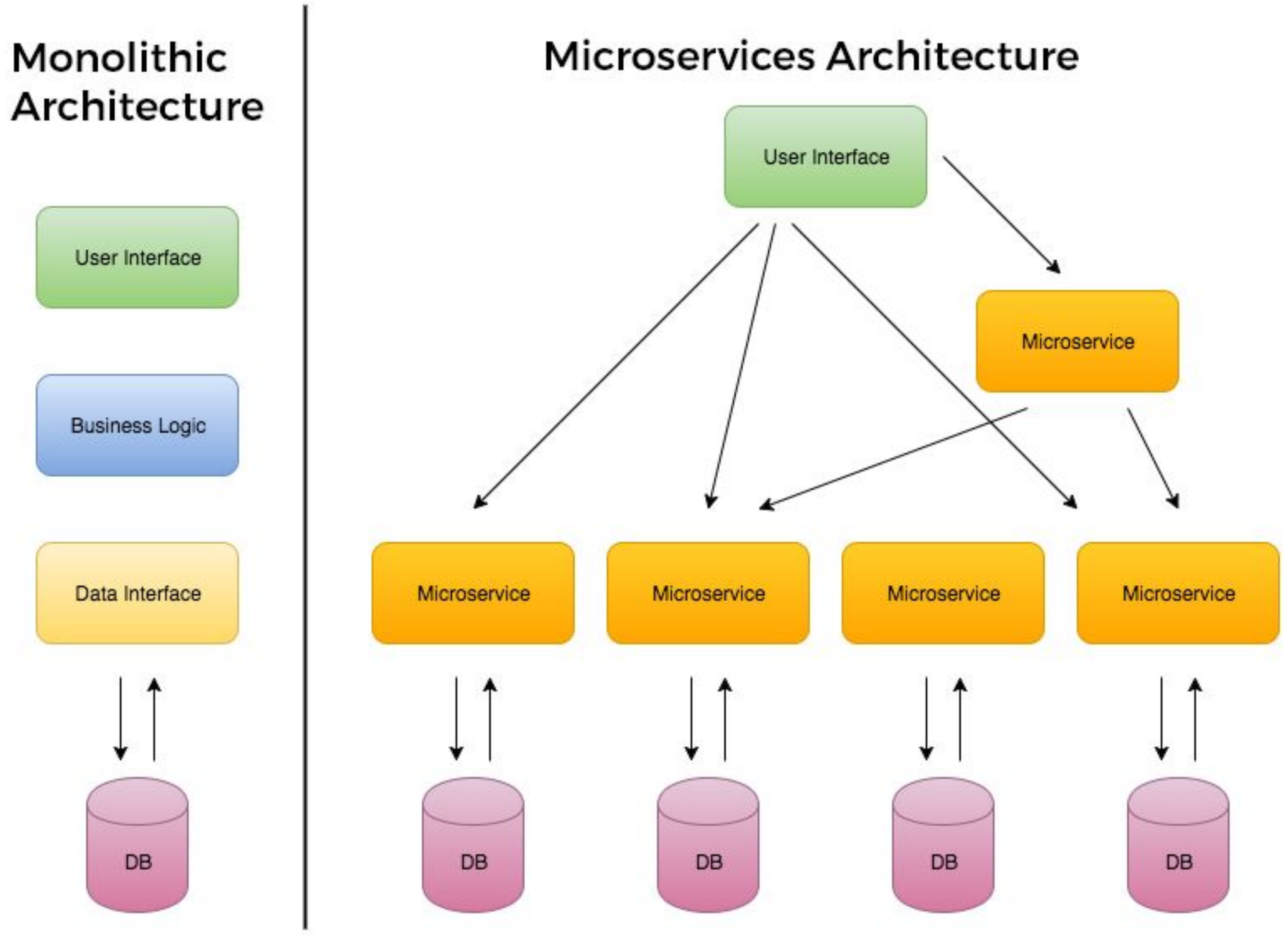
- Logstash supports all the mainstream log types, diverse plug-ins, and flexible customization, but has relatively low performance and is prone to high memory usage because of JVM.
- Fluentd supports all the mainstream log types and many plug-ins, and delivers good performance.
- Fluent bit doesn’t has small number of plugins but is very small and has good performance
- Filebeat - small but has many issues about memory leaks!!!

Distributed tracing systems

Monolithic service architectures for large backend applications are becoming increasingly rare. The monoliths are being replaced with distributed system architectures, where the backend application is spread out (distributed) in an ecosystem of small and narrowly-focused services. These distributed services communicate with each other over the network to process requests.

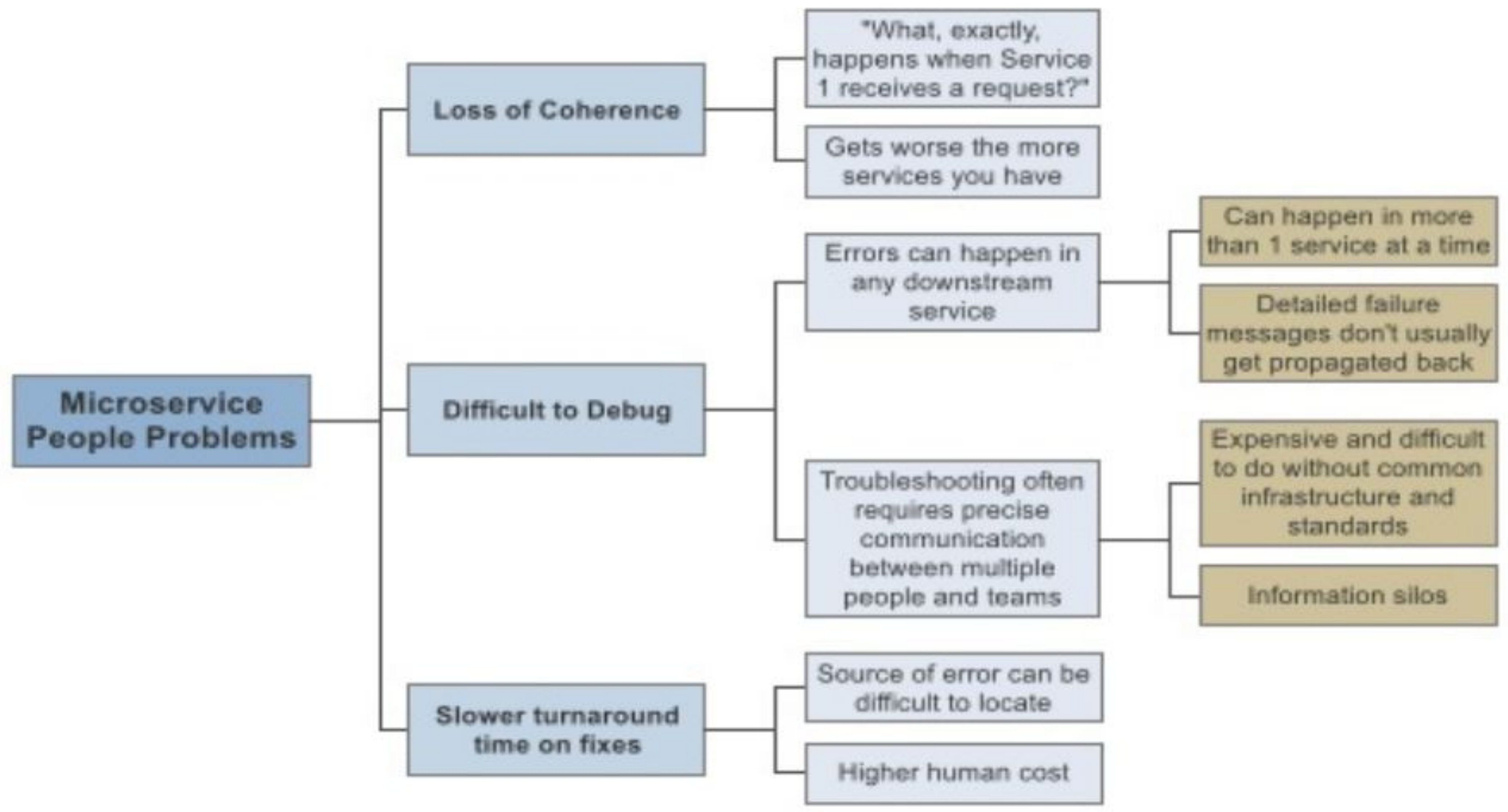
There are many benefits to distributed system architectures, but we’re not trying to sell you on them in this blog post. A quick microservice vs. monolith Google search can do that. Instead, we’re going to focus on the difficulty of tracking and analyzing requests in a distributed environment; pinpointing problems can be very frustrating when requests might touch dozens of services during processing.

Distributed tracing systems



Distributed tracing systems

Microservice People Problems


Microservice People Problems
Loss of Coherence
"What, exactly, happens when Service 1 receives a request?"
Gets worse the more services you have
Difficult to Debug
Errors can happen in any downstream service
Can happen in more than 1 service at a time
Detailed failure messages don't usually get propagated back
Troubleshooting often requires precise communication between multiple people and teams
Expensive and difficult to do without common infrastructure and standards
Information silos
Slower turnaround time on fixes
Source of error can be difficult to locate
Higher human cost

Distributed tracing systems

Open source distributed tracing tools:

1. Zipkin - Twitter. Now - CNCF
2. Jaeger - Uber. Now - CNCF
3. Opencensus - Google
4. Opentracing - CNCF

Distributed tracing systems

Zipkin and Jaeger are two popular choices for request tracing. Zipkin was originally inspired by Dapper and developed by Twitter. It's now maintained by a dedicated community.

Jaeger was originally built and open sourced by Uber.

Jaeger is a Cloud Native Computing Foundation project. The overall architecture is similar. Instrumented systems send events/traces to the trace collector.

Distributed tracing systems

OpenTracing is a spec that grew out of Zipkin to provide cross-platform compatibility. It offers a vendor-neutral API for adding tracing to applications and delivering that data into distributed tracing systems. A library written for the OpenTracing spec can be used with any system that is OpenTracing-compliant.

OpenCensus is a vendor-agnostic (unified framework) single distribution of libraries to provide **metrics** collection and **tracing** for your services. *OpenCensus* originates from *Google*.

Distributed tracing systems: Zipkin vs Jaeger

	JAEGER	**ZIPKIN**
OpenTracing compatibility	Yes	Yes
Language support	Python Go Node Python Java C++ C# Ruby PHP	Go Java JavaScript Ruby Scala C++ C# Python PHP
Storage support	In-memory Cassandra Elasticsearch ScyllaDB (work in progress)	In-memory Cassandra Elasticsearch Mysql

Distributed tracing systems: Zipkin vs Jaeger

	JAEGER	**ZIPKIN**
Sampling	Dynamic sampling rate (supports rate limiting and probabilistic sampling strategies)	Fixed sampling rate (supports probabilistic sampling strategy)
SPAN transport	UDP HTTP	HTTP Kafka Scribe AMQP
Docker ready	Yes	Yes
Jaeger Agent in Kubernetes as DaemonSet	Yes	No

Distributed tracing systems: SPAN

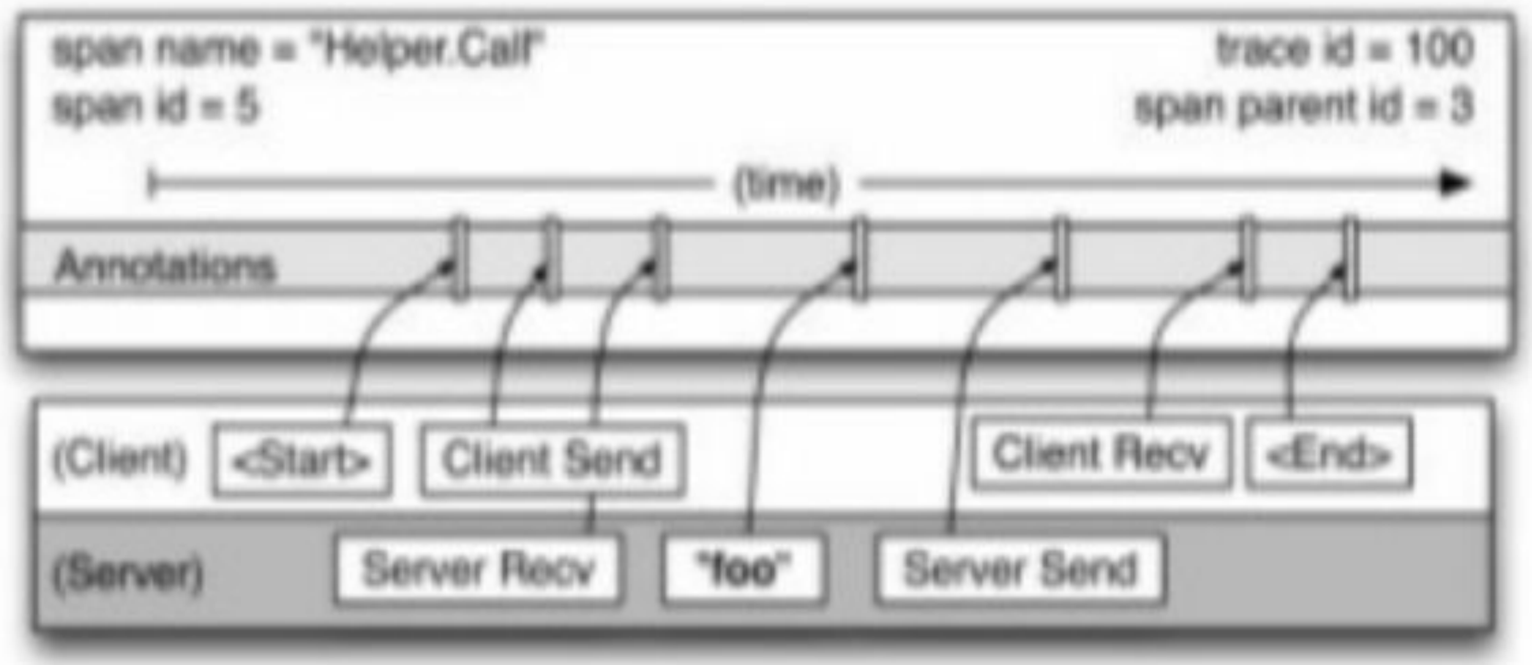
A set of Annotations and BinaryAnnotations that correspond to a particular RPC. Spans contain identifying information such as traceId, spanId, parentId, and RPC name.

Spans are usually small. For example, the serialized form is often measured in KiB or less. When spans grow beyond orders of KiB, other problems occur, such as hitting limits like Kafka message size (1MiB). Even if you can raise message limits, large spans will increase the cost and decrease the usability of the tracing system. For this reason, be conscious to store data that helps explain system behavior, and don’t store data that doesn’t.

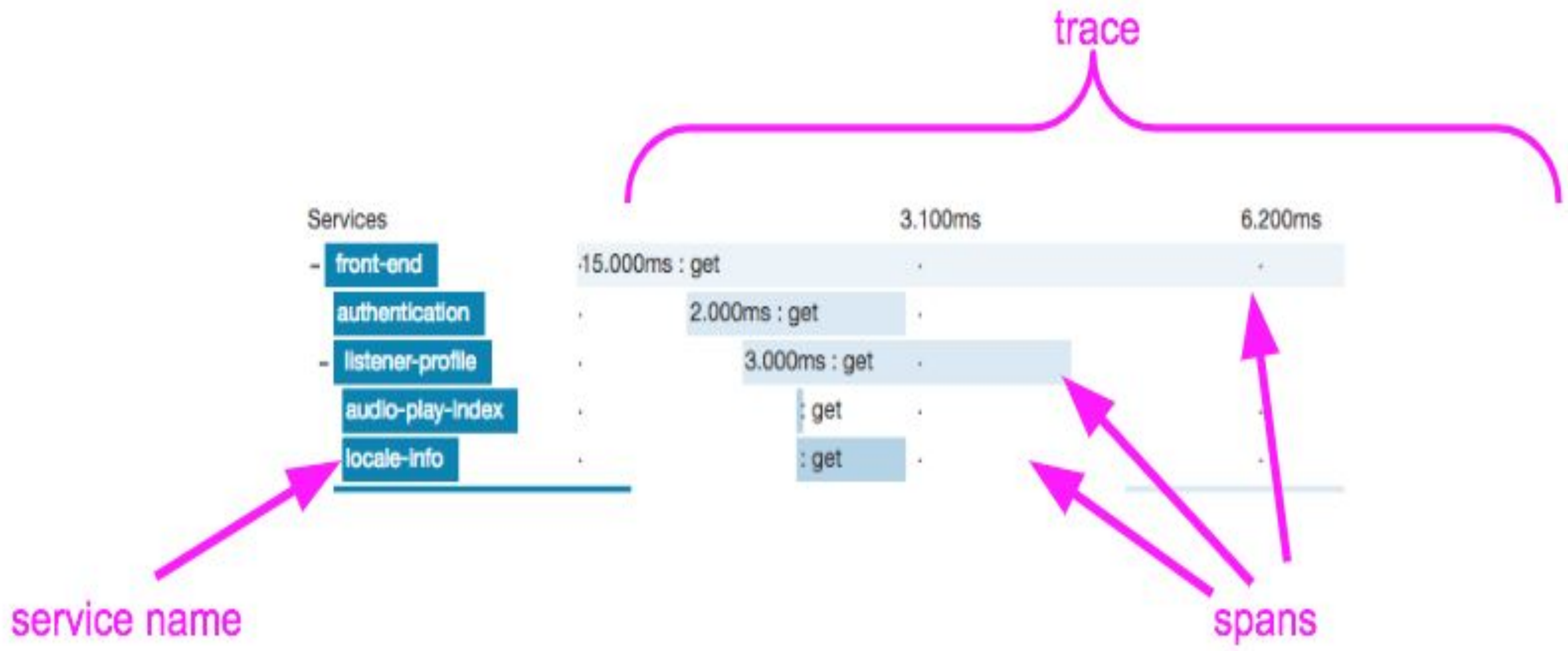
Distributed tracing systems: SPAN

Logged event in a typical span

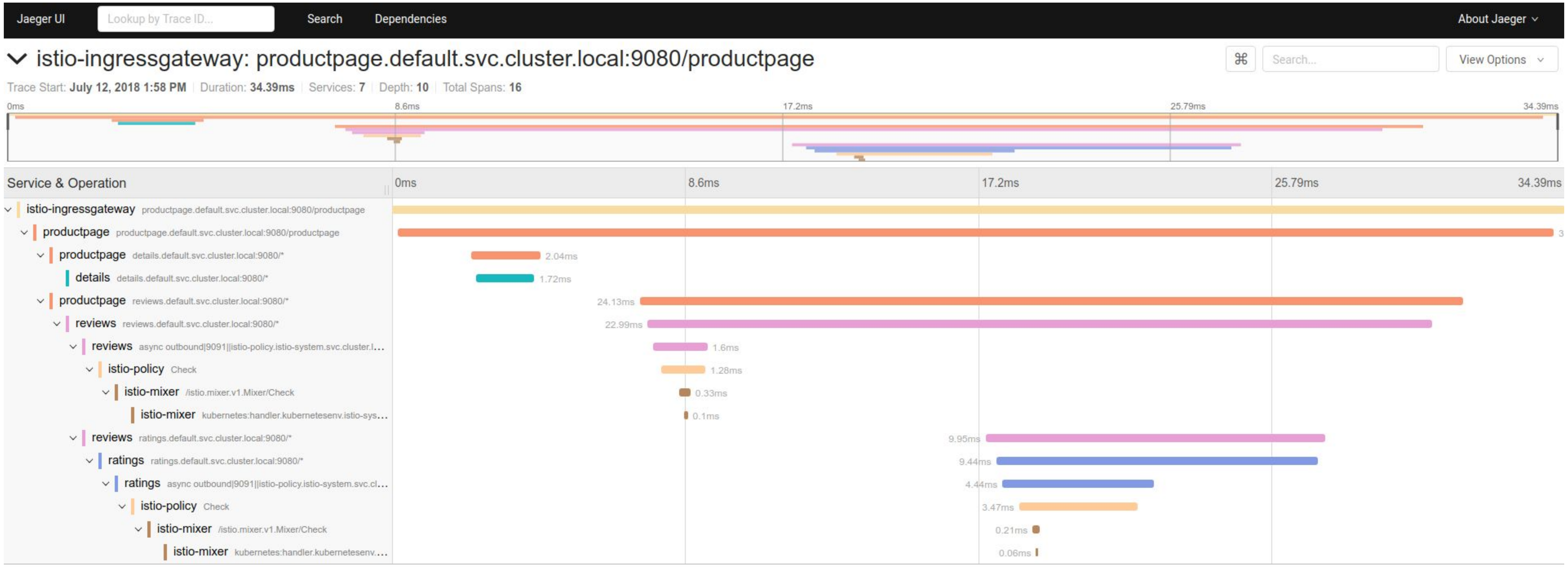
- Span name
- Span start time
- Span end time
- Trace id
- Span id
- Span parent id
- Any timing information recorded by the instrumentation library (RPC, HTTP)
- Additional custom labels ("foo")



Distributed tracing systems: SPAN



Distributed tracing systems



Jaeger UI | Lookup by Trace ID... | Search | Dependencies | About Jaeger

istio-ingressgateway: productpage.default.svc.cluster.local:9080/productpage

Trace Start: **July 12, 2018 1:58 PM** | Duration: **34.39ms** | Services: **7** | Depth: **10** | Total Spans: **16**

Service & Operation	
istio-ingressgateway productpage.default.svc.cluster.local:9080/productpage	
productpage productpage.default.svc.cluster.local:9080/productpage	3
productpage details.default.svc.cluster.local:9080/*	2.04ms
details details.default.svc.cluster.local:9080/*	1.72ms
productpage reviews.default.svc.cluster.local:9080/*	24.13ms
reviews reviews.default.svc.cluster.local:9080/*	22.99ms
reviews async outbound|9091||istio-policy.istio-system.svc.cluster.l…	1.6ms
istio-policy Check	1.28ms
istio-mixer /istio.mixer.v1.Mixer/Check	0.33ms
istio-mixer kubernetes:handler.kubernetesenv.istio-sys…	0.1ms
reviews ratings.default.svc.cluster.local:9080/*	9.95ms
ratings ratings.default.svc.cluster.local:9080/*	9.44ms
ratings async outbound|9091||istio-policy.istio-system.svc.cl…	4.44ms
istio-policy Check	3.47ms
istio-mixer /istio.mixer.v1.Mixer/Check	0.21ms
istio-mixer kubernetes:handler.kubernetesenv.…	0.06ms

Distributed tracing systems: Opencensus

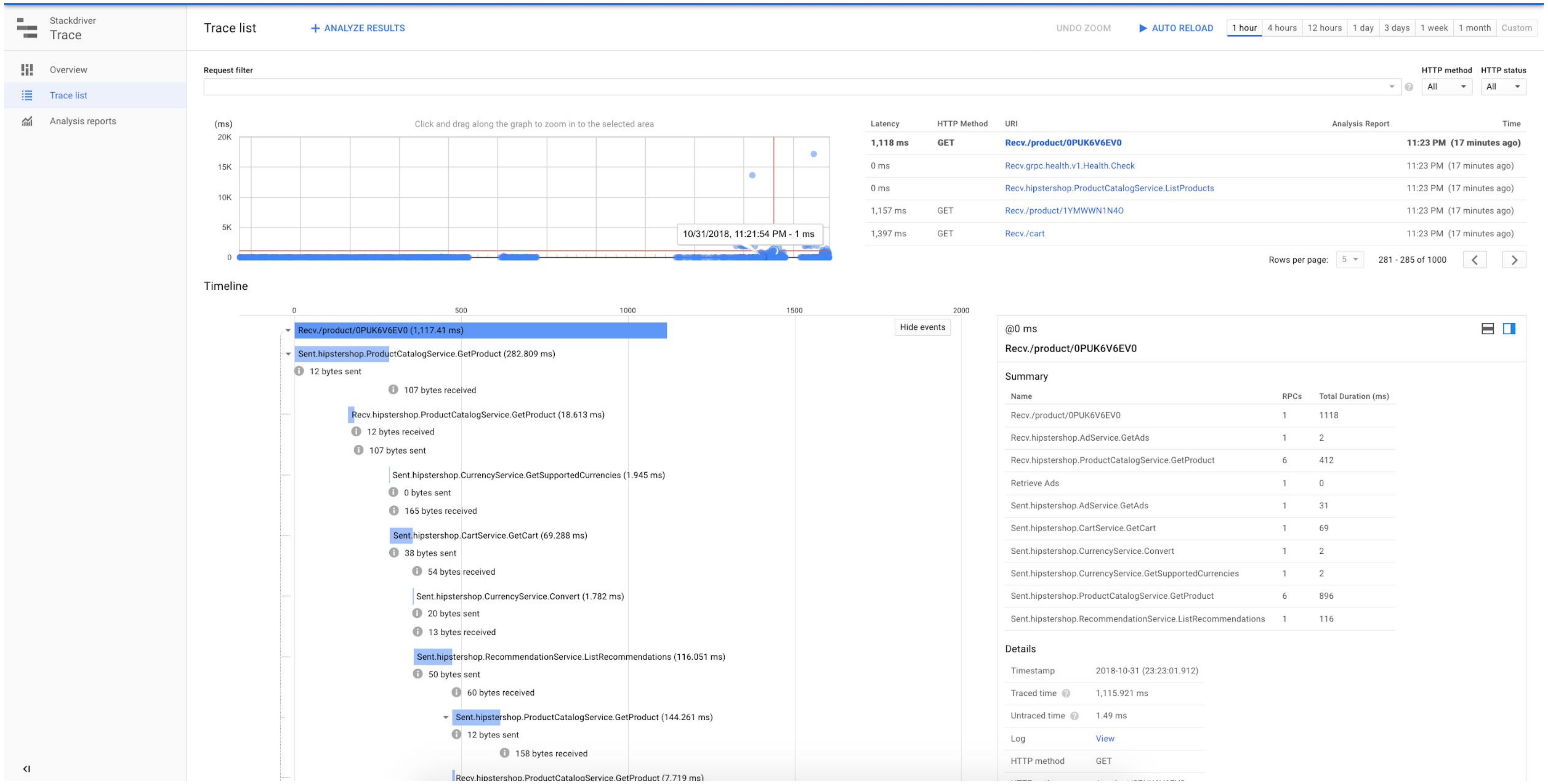
OpenCensus is an open source library for distributed tracing and monitoring. There is overlap between the OpenCensus APIs and the Stackdriver APIs, so that an alternative would be to develop the probe with Stackdriver APIs. However, there are several advantages of using OpenCensus instead:

1. You avoid an explicit dependency on a proprietary API. This will be important to people developing open source projects and companies wanting vendor independence. See the GitHub repository for the code.
2. OpenCensus has additional features for metric aggregation, which can intelligently help reduce the volume of monitoring data and controlling monitoring costs. See the Aggregation type for details on how to customize this.

Distributed tracing systems: Opencensus

3. OpenCensus supports exporters to multiple backends. That enables it development of code for collectors that can export monitoring stats to multiple clouds. This post uses the Stackdriver exporter to send the data to Google Cloud. See the OpenCensus documentation for other exporters.
4. OpenCensus supports has prebuilt instrumentation for native library features, such as HTTP and gRPC. The code for this post uses the OpenCensus HTTP plugin, which reduces the amount of code that needs to be written.

Distributed tracing systems: Opencensus



Latency	HTTP Method	URI	Analysis Report	Time
1,118 ms	**GET**	**Recv./product/0PUK6V6EV0**		**11:23 PM (17 minutes ago)**
0 ms		Recv.grpc.health.v1.Health.Check		11:23 PM (17 minutes ago)
0 ms		Recv.hipstershop.ProductCatalogService.ListProducts		11:23 PM (17 minutes ago)
1,157 ms	GET	Recv./product/1YMWWN1N40		11:23 PM (17 minutes ago)
1,397 ms	GET	Recv./cart		11:23 PM (17 minutes ago)

Rows per page: 5 281 - 285 of 1000

```
Recv./product/0PUK6V6EV0 (1,117.41 ms)
Sent.hipstershop.ProductCatalogService.GetProduct (282.809 ms)
  12 bytes sent
  107 bytes received
Recv.hipstershop.ProductCatalogService.GetProduct (18.613 ms)
  12 bytes received
  107 bytes sent
Sent.hipstershop.CurrencyService.GetSupportedCurrencies (1.945 ms)
  0 bytes sent
  165 bytes received
Sent.hipstershop.CartService.GetCart (69.288 ms)
  38 bytes sent
  54 bytes received
Sent.hipstershop.CurrencyService.Convert (1.782 ms)
  20 bytes sent
  13 bytes received
Sent.hipstershop.RecommendationService.ListRecommendations (116.051 ms)
  50 bytes sent
  60 bytes received
Sent.hipstershop.ProductCatalogService.GetProduct (144.261 ms)
  12 bytes sent
  158 bytes received
Recv.hipstershop.ProductCatalogService.GetProduct (7.719 ms)
```

@0 ms

Recv./product/0PUK6V6EV0

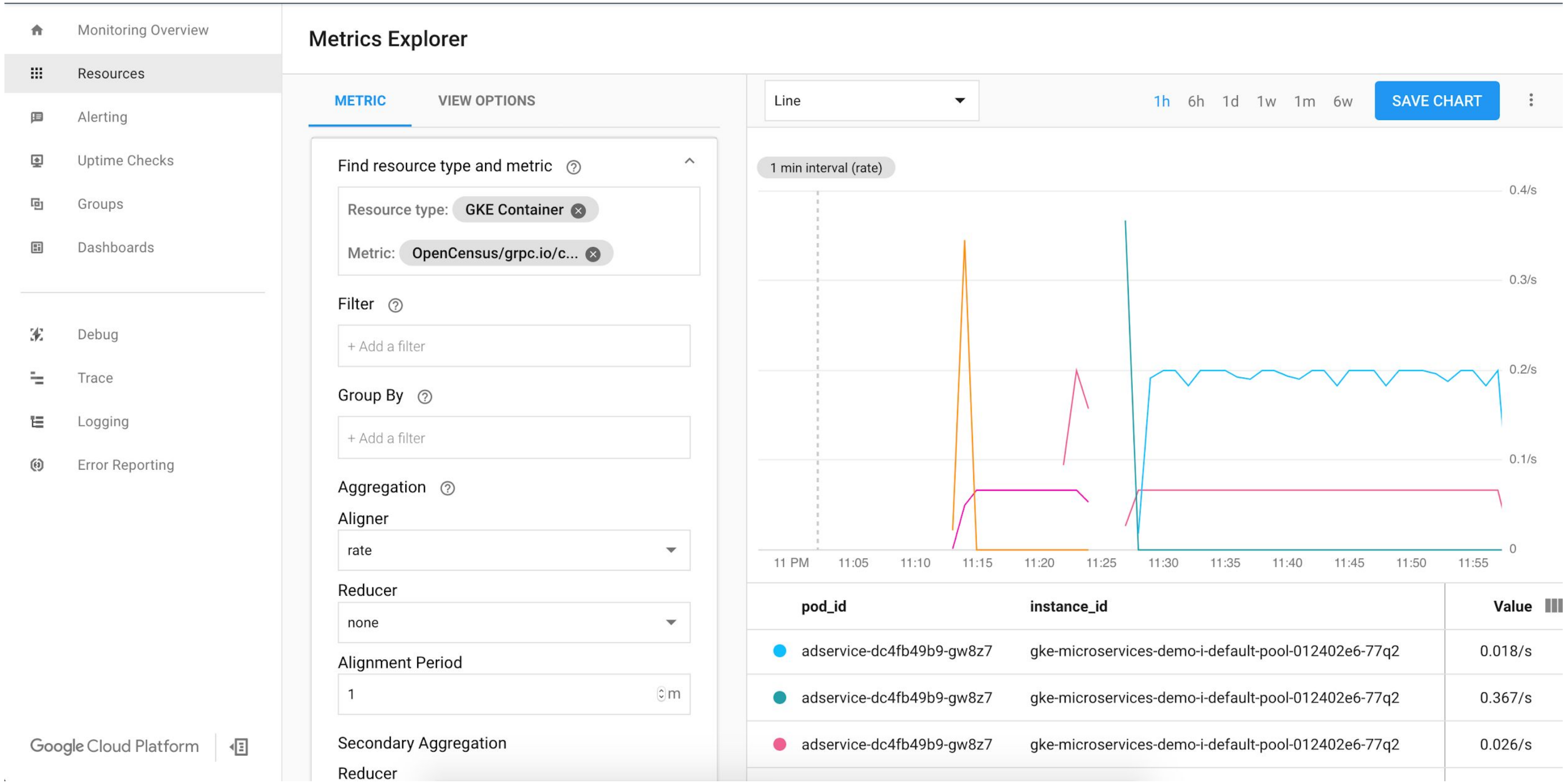
Summary

Name	RPCs	Total Duration (ms)
Recv./product/0PUK6V6EV0	1	1118
Recv.hipstershop.AdService.GetAds	1	2
Recv.hipstershop.ProductCatalogService.GetProduct	6	412
Retrieve Ads	1	0
Sent.hipstershop.AdService.GetAds	1	31
Sent.hipstershop.CartService.GetCart	1	69
Sent.hipstershop.CurrencyService.Convert	1	2
Sent.hipstershop.CurrencyService.GetSupportedCurrencies	1	2
Sent.hipstershop.ProductCatalogService.GetProduct	6	896
Sent.hipstershop.RecommendationService.ListRecommendations	1	116

Details

Timestamp	2018-10-31 (23:23:01.912)
Traced time	1,115.921 ms
Untraced time	1.49 ms
Log	View
HTTP method	GET

Distributed tracing systems: Opencensus



Find resource type and metric

Resource type: GKE Container

Metric: OpenCensus/grpc.io/c...

Filter

+ Add a filter

Group By

+ Add a filter

Aggregation

Aligner: rate

Reducer: none

Alignment Period: 1 m

Secondary Aggregation

Reducer

	pod_id	instance_id	Value
●	adservice-dc4fb49b9-gw8z7	gke-microservices-demo-i-default-pool-012402e6-77q2	0.018/s
●	adservice-dc4fb49b9-gw8z7	gke-microservices-demo-i-default-pool-012402e6-77q2	0.367/s
●	adservice-dc4fb49b9-gw8z7	gke-microservices-demo-i-default-pool-012402e6-77q2	0.026/s

Distributed tracing systems: Conclusion

In the end, how important each piece is depends heavily on who you are and what kind of system you are building. For example, open source library authors are very interested in the OpenTracing API, while service developers tend to be more interested in the Trace-Context specification. When someone says one piece is more important than the other, they usually mean “one piece is more important to *me* than the other."

However, the reality is this: Distributed Tracing has become a necessity for monitoring modern systems. In designing the building blocks for these systems, the age-old approach—"decouple where you can"—still holds true. Cleanly decoupled components are the best way to maintain flexibility and forwards-compatibility when building a system as cross-cutting as a distributed monitoring system.

Error tracking tools

We all want our application to run as smoothly as possible, but that’s not always the case. Once the application is up and running, we need to know if and when errors or exceptions are thrown. That’s why there are numerous error tracking tools in the market.

What is an error tracking tool? What can you actually do with it? What are the differences between the tools? Answers to these questions, along with some other relevant questions are waiting for you in the following post. Get ready to squash some bugs.

Error tracking tools

Deploying fast and frequently might introduce a errors, exceptions and bugs into our application, and it’s best to identify them before your users do.

Current error tracking tools:

1. Sentry
2. Airbrake
3. BugSnag
4. OverOps
5. StackHunter
6. Rollbar
7. Raygun

Error tracking tools

Sentry a tiny bit of open source code grew to became a full-blown error monitoring tool, that identifies and debugs errors in production. You can use the insights to know whether you should spend time fixing a certain bug, or if you should roll back to a previous version.

The dashboard lets you see stack traces, with support for source maps, along with detecting each error’s URL, parameters and session information. Each trace can be filtered with app, framework or raw error views.

Supported languages:

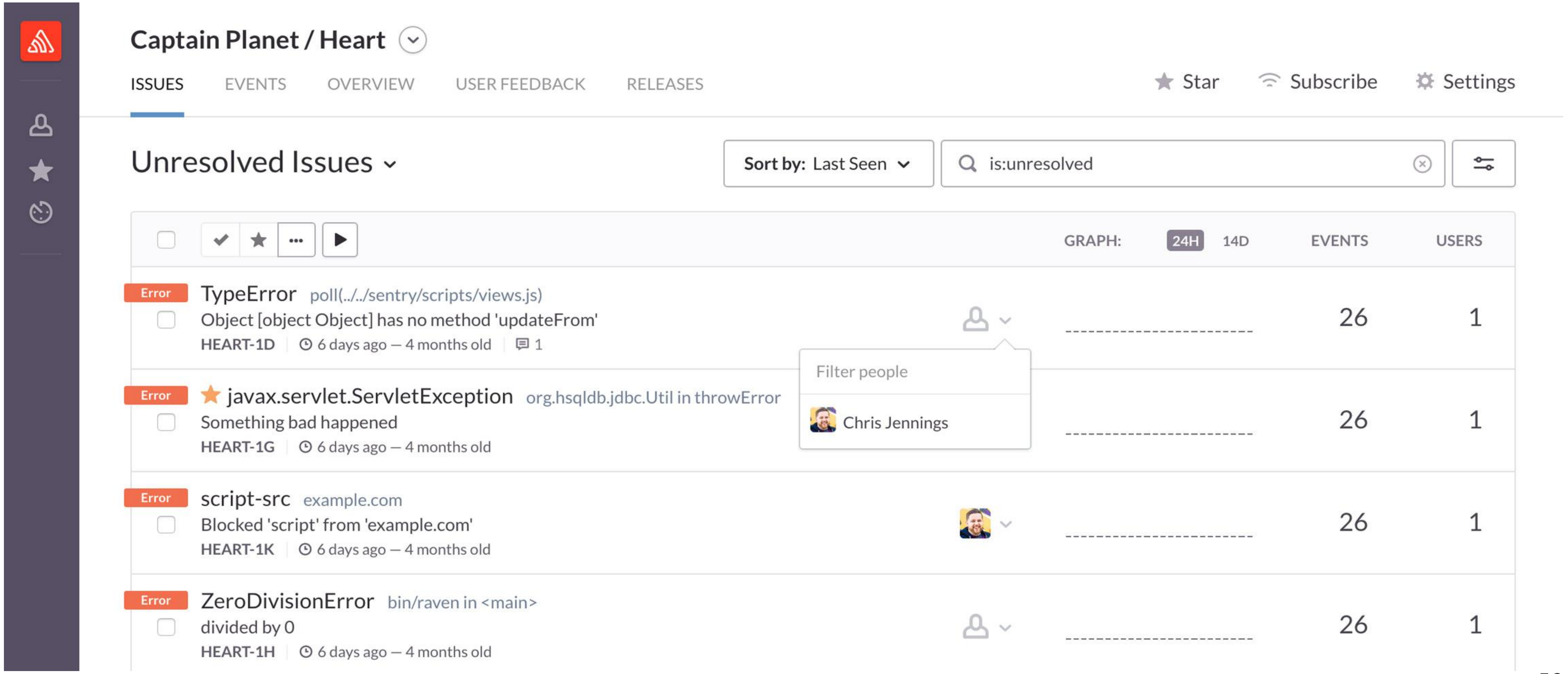
JavaScript, Node.js, Python, Go, PHP, Ruby, C#, Objective-C and even Swift

Error tracking tools

Features

You can catch errors in real-time as you deploy, and every error includes information about software, environment and the users themselves. If you believe your users are a key point at understanding errors, Sentry lets you prompt users for feedback whenever they encounter errors. That allows comparing their individual experience to the data you already have.

Error tracking tools


Captain Planet / Heart

ISSUES EVENTS OVERVIEW USER FEEDBACK RELEASES Star Subscribe Settings

Unresolved Issues

Sort by: Last Seen | is:unresolved

GRAPH: 24H 14D EVENTS USERS

Error TypeError poll(../../sentry/scripts/views.js)
Object [object Object] has no method 'updateFrom'
HEART-1D | 6 days ago — 4 months old | 1 — 26 — 1

Filter people
Chris Jennings

Error javax.servlet.ServletException org.hsqldb.jdbc.Util in throwError
Something bad happened
HEART-1G | 6 days ago — 4 months old — 26 — 1

Error script-src example.com
Blocked 'script' from 'example.com'
HEART-1K | 6 days ago — 4 months old — 26 — 1

Error ZeroDivisionError bin/raven in <main>
divided by 0
HEART-1H | 6 days ago — 4 months old — 26 — 1

GCP Stackdriver

Stackdriver Logging provides you with the ability to filter, search, and view logs from your cloud and open source application services. Allows you to define metrics based on log contents that are incorporated into dashboards and alerts.

GCP Stackdriver: Features

1. Debugger
2. Alert
3. Error reporting
4. Tracing
5. Rapid discovery
6. Logging
7. Uptime monitoring
8. Dashboard
9. Integrations
10. Profiling
11. Smart defaults

Conclusion

Keeping in mind that you need to be proactive in your monitoring (i.e., prevent problems before they reach users), you need a solution that can test for performance issues in your pre-production environments and baseline the performance of your production environment. This means you’ll want to opt for a synthetic monitoring solution over real user monitoring. Remember that with synthetic monitoring, you’ll be developing behavioral scripts to simulate user actions and your solution will monitor them at specified intervals for performance, including availability, functionality, and response times.

Thank you for your attention!

Questions?

CONTACTS:

Email: stas.kolenkin@gmail.com

Skype: stas.kolenkin