# Managing Observability in Modern Applications

Ran Ribenzaft

@ranrib

# > whoami

- # CTO @ Epsagon
- # AWS Serverless Hero
- <------ Looking for whales in Hawaii 🐳
- 🐦 @ranrib

# Epsagon

An **automated & agentless Observability** solution, built for **microservices in any cloud**

# What we'll discuss today

- Monitoring and Logging
- Observability
- Distributed Tracing

# Why monitoring?

## Make sure our business works

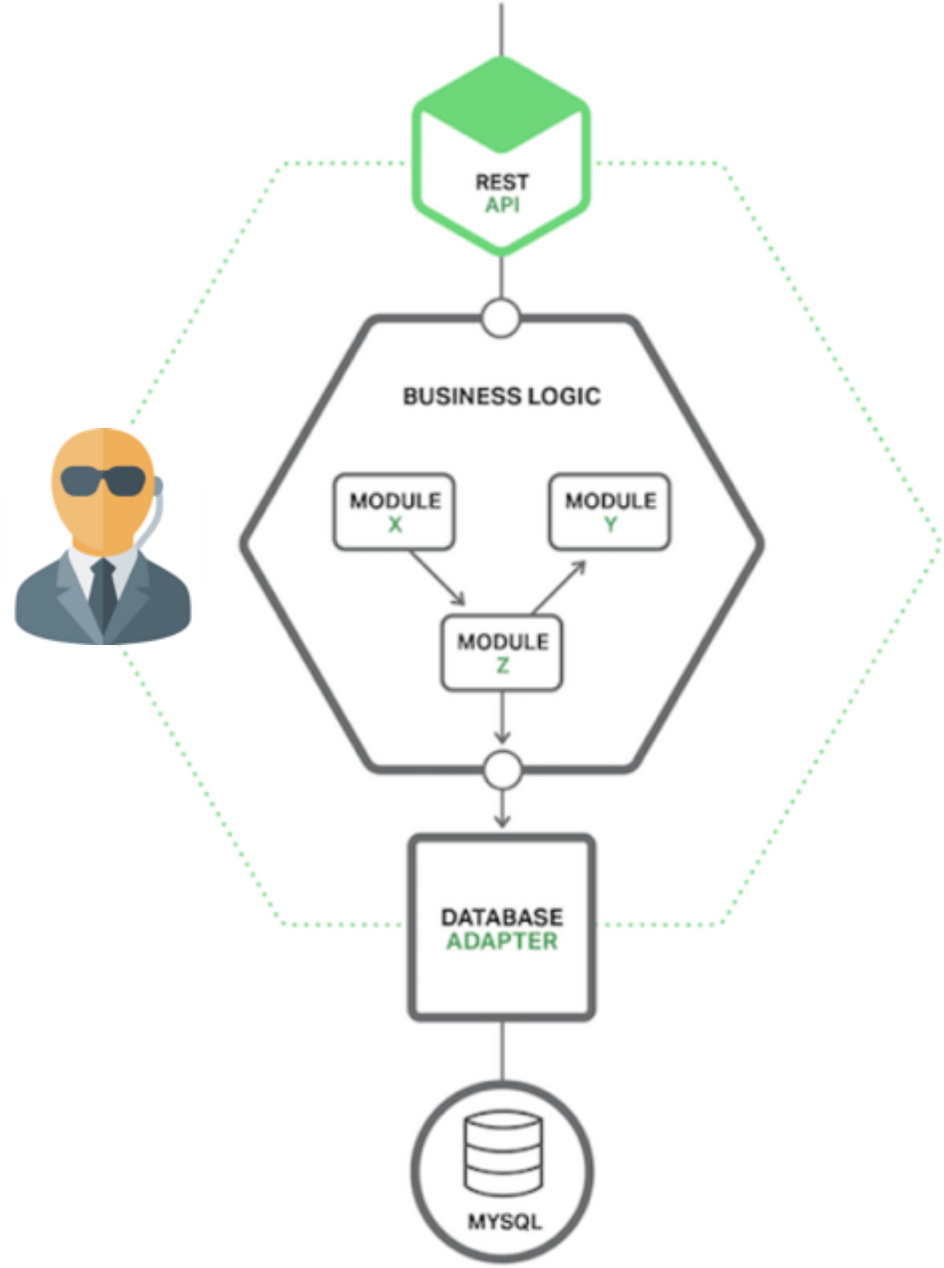
# What should we monitor?

- 4 golden signals from Google’s SRE book

- Latency
- Traffic
- Errors
- Saturation

# Old school monitoring

- Agent based
- Collects only host data
- Collects only metrics

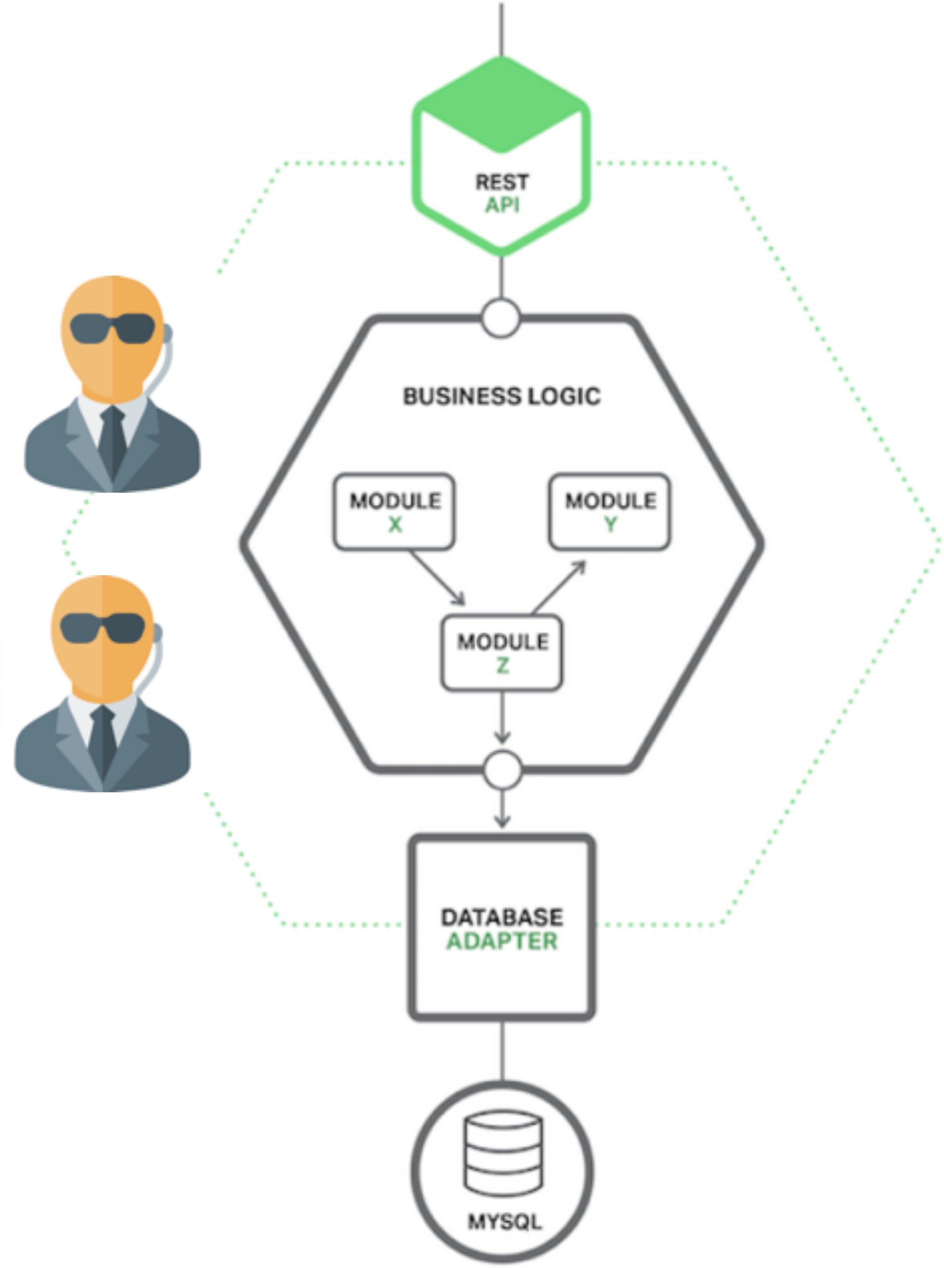


# Troubleshooting

We need more debug data -> logs

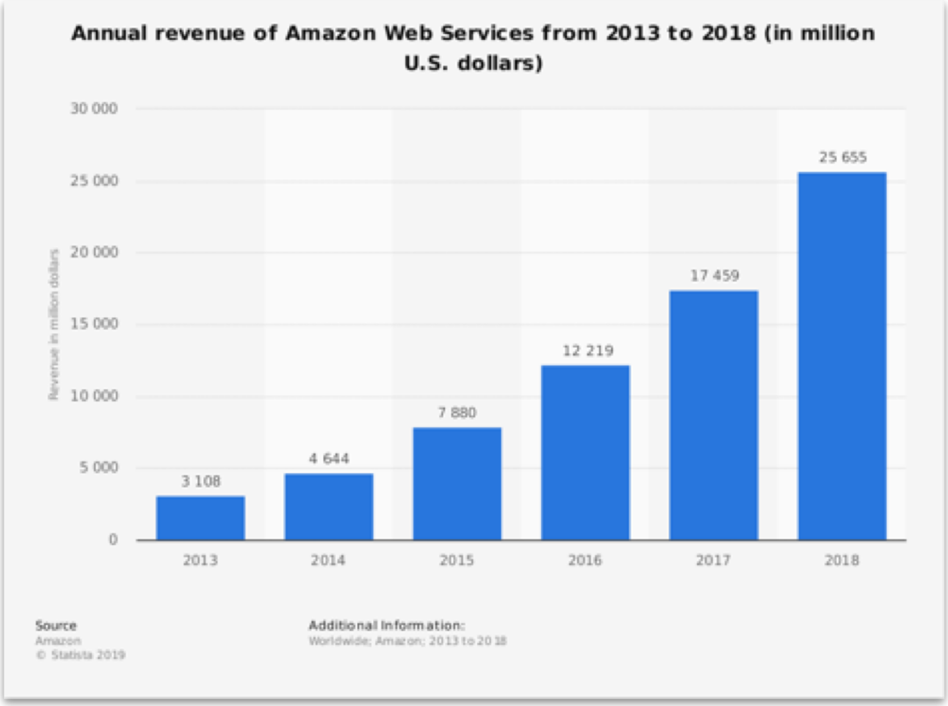
# Old school logging

- Agent based
- Dumps locally or remotely
- Collects only logged data



# Fast forward into the future

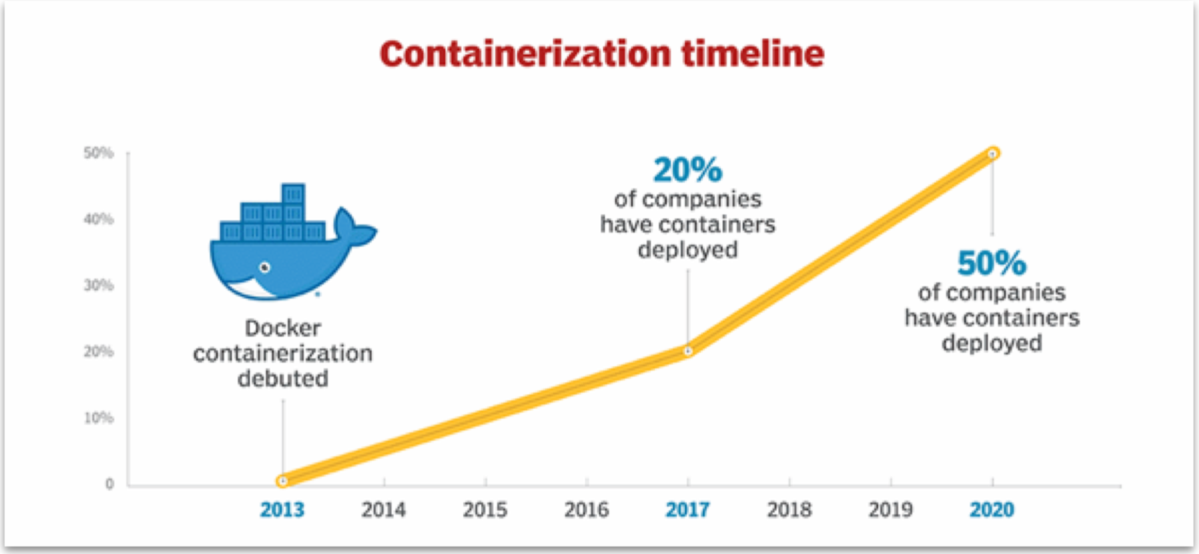
# Fast-Growing Market: Cloud + Microservices



Annual revenue of Amazon Web Services from 2013 to 2018 (in million U.S. dollars)

Source
Amazon
© Statista 2019

Additional Information:
Worldwide; Amazon; 2013 to 2018

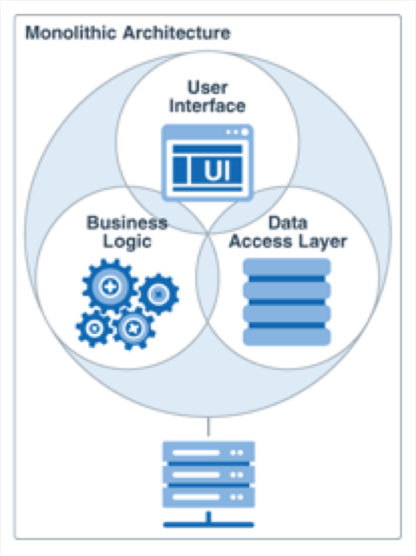


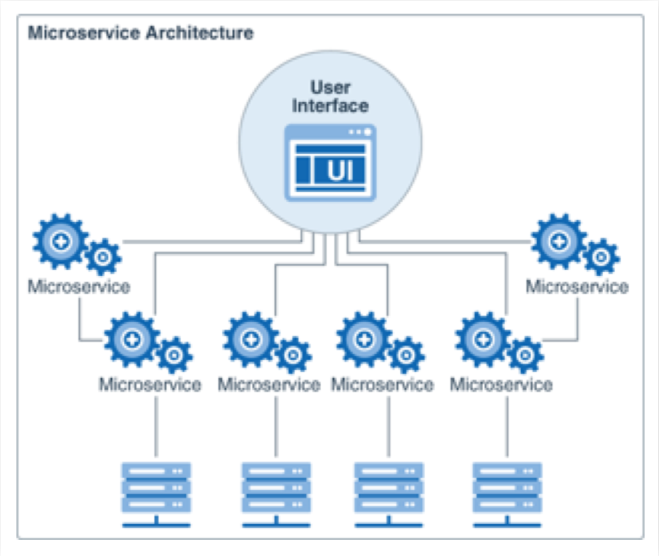
Containerization timeline

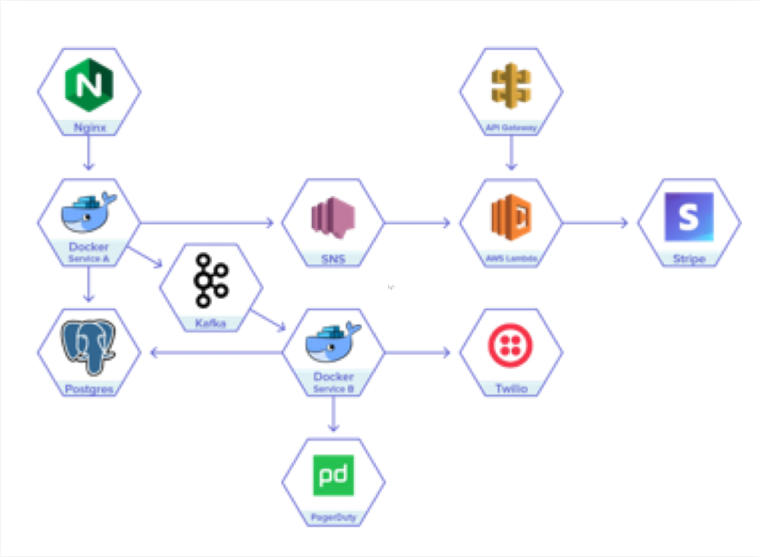
epsagon

@ranrib

# The Rise of Microservices on the Cloud







Host-based
Monolithic

→

Host-based
Distributed

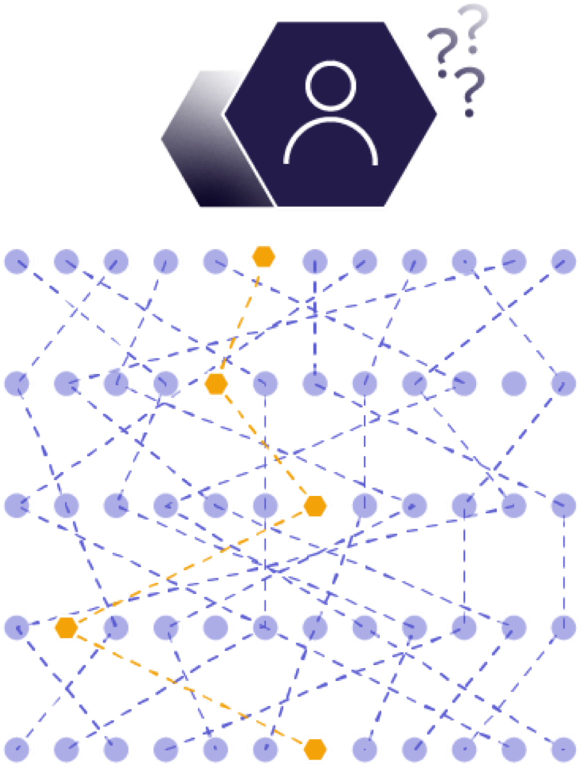
→

Abstracted host
Highly distributed

**Extremely hard to monitor and troubleshoot!**

# Challenges for Engineering and DevOps

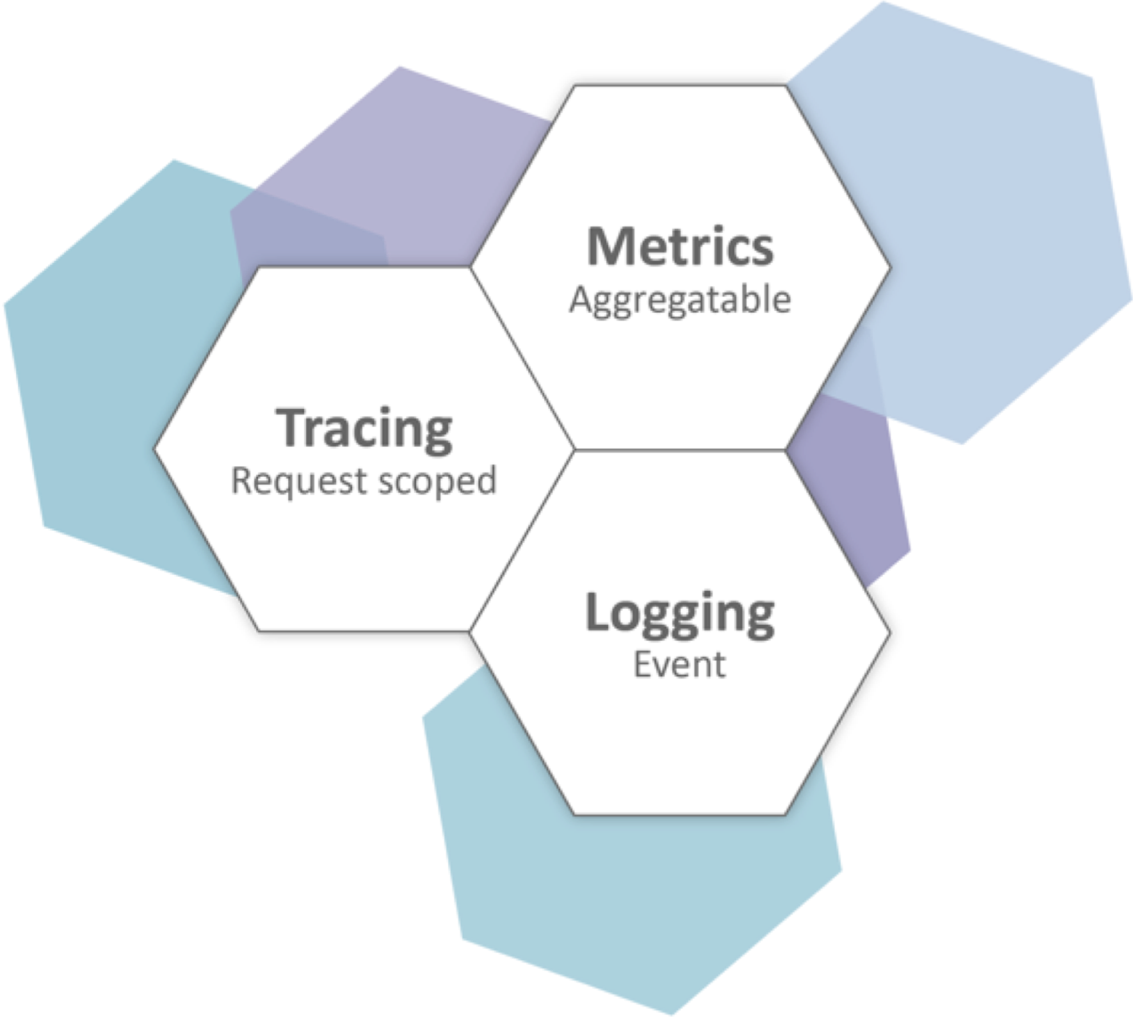
- **Troubleshooting**
  Are basic logs and metrics the right tool for highly distributed applications?
- **Monitoring**
  “Is my application working properly”?
- **Development**
  I’m not sure what’s currently running in production.
  How can I build new services?



epsagon

@ranrib

# The Three Pillars of Observability



# Monitoring best practices

- Aggregate all metrics into a unified dashboard
- Define your critical metrics (thresholds)
- Use custom business metrics

# Monitoring best practices

- Monitor application metrics:

- Avg. duration of calls to an HTTP API
- Minimum number of calls to a message queue
- Number of 500/400 errors

# Logging best practices

- Print out JSONed logs with metadata (service name, stage, etc.)
- Automate the process of logging
- Index the fields you're are using

# Something is still missing

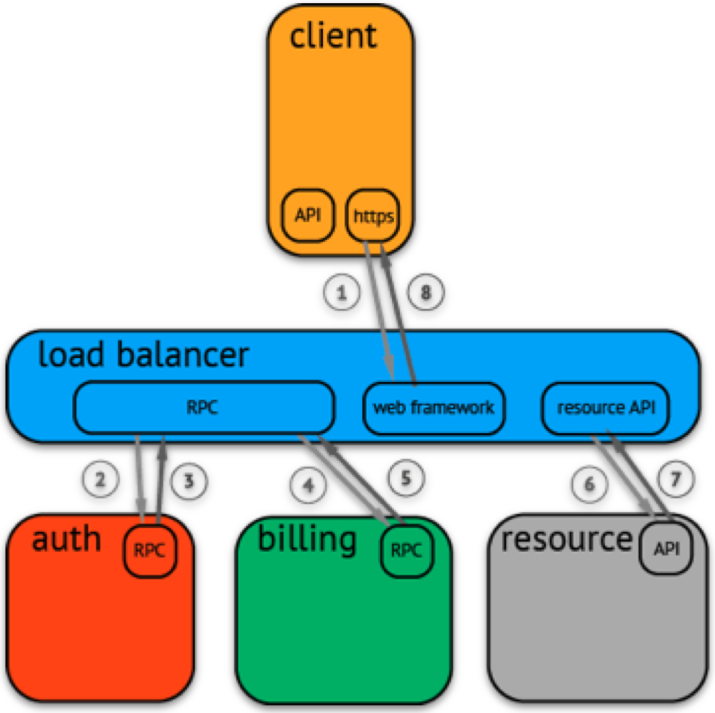
- How do we correlate between metrics and logs
- How do we correlate between data in different services

epsagon

# Distributed tracing

# Distributed tracing

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



epsagon

@ranrib

# Distributed tracing

- Generating traces

OPENTRACING

OpenTelemetry

- Ingestion and client

JAEGER

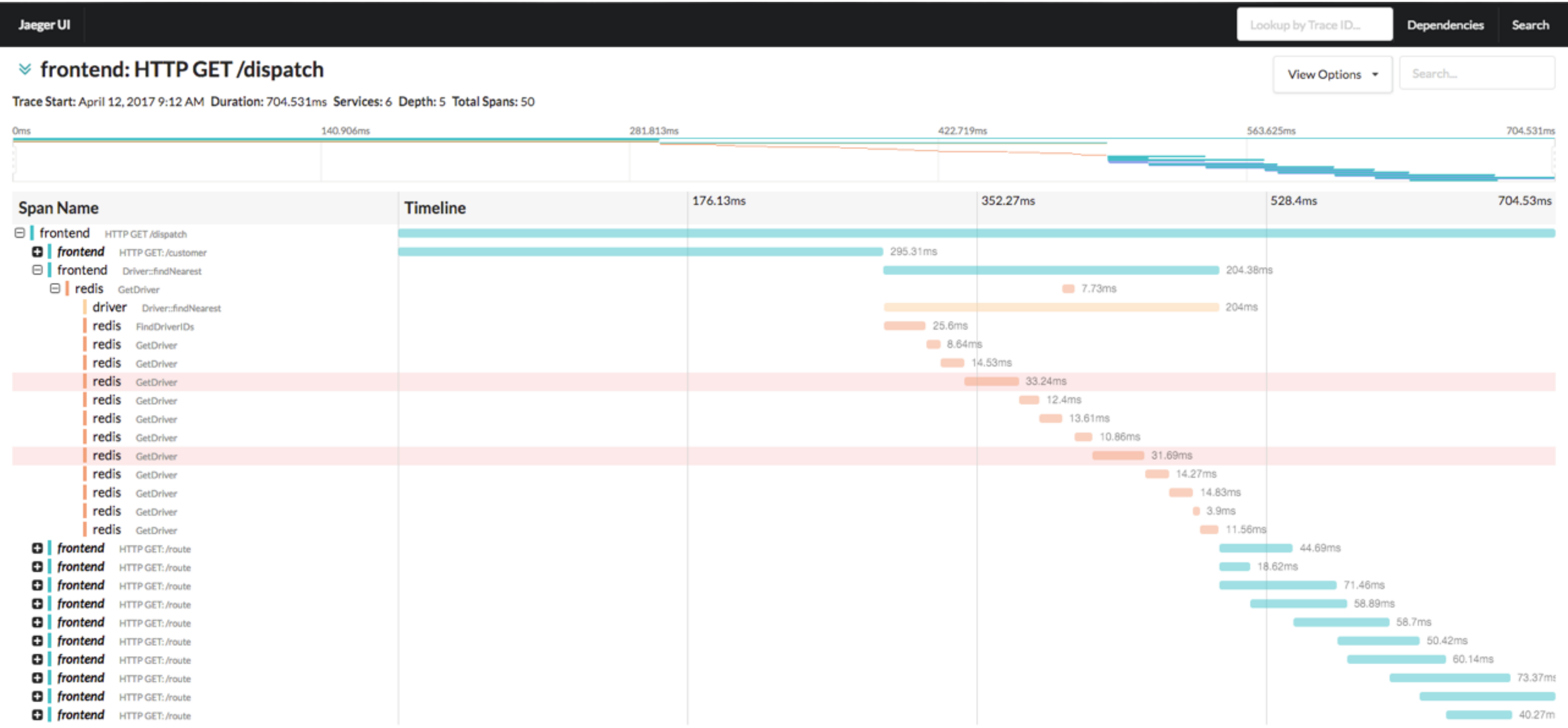
ZIPKIN

# Generating traces

- **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

# Ingestion and client

- Ingestion according to our scale (millions? billions?)
- Index context and tags for easy search
- Visualize traces (timeline, graph)
- Set alerts
- …



## frontend: HTTP GET /dispatch

**Trace Start:** April 12, 2017 9:12 AM **Duration:** 704.531ms **Services:** 6 **Depth:** 5 **Total Spans:** 50

| Span Name | | Timeline |
|---|---|---|
| frontend | HTTP GET /dispatch | |
| frontend | HTTP GET: /customer | 295.31ms |
| frontend | Driver::findNearest | 204.38ms |
| redis | GetDriver | 7.73ms |
| driver | Driver::findNearest | 204ms |
| redis | FindDriverIDs | 25.6ms |
| redis | GetDriver | 8.64ms |
| redis | GetDriver | 14.53ms |
| redis | GetDriver | 33.24ms |
| redis | GetDriver | 12.4ms |
| redis | GetDriver | 13.61ms |
| redis | GetDriver | 10.86ms |
| redis | GetDriver | 31.69ms |
| redis | GetDriver | 14.27ms |
| redis | GetDriver | 14.83ms |
| redis | GetDriver | 3.9ms |
| redis | GetDriver | 11.56ms |
| frontend | HTTP GET: /route | 44.69ms |
| frontend | HTTP GET: /route | 18.62ms |
| frontend | HTTP GET: /route | 71.46ms |
| frontend | HTTP GET: /route | 58.89ms |
| frontend | HTTP GET: /route | 58.7ms |
| frontend | HTTP GET: /route | 50.42ms |
| frontend | HTTP GET: /route | 60.14ms |
| frontend | HTTP GET: /route | 73.37ms |
| frontend | HTTP GET: /route | |
| frontend | HTTP GET: /route | 40.27m |

# Tagging traces

- Adding tags for search and aggregations

- Identifiers – user_id
- Flow control – event_type
- Business metrics – items_in_cart

# Tracing with payload

- Search an event according to:

- user_id (from HTTP headers)
- key in NoSQL
- Response payload from HTTP call

# Tracing as a glue

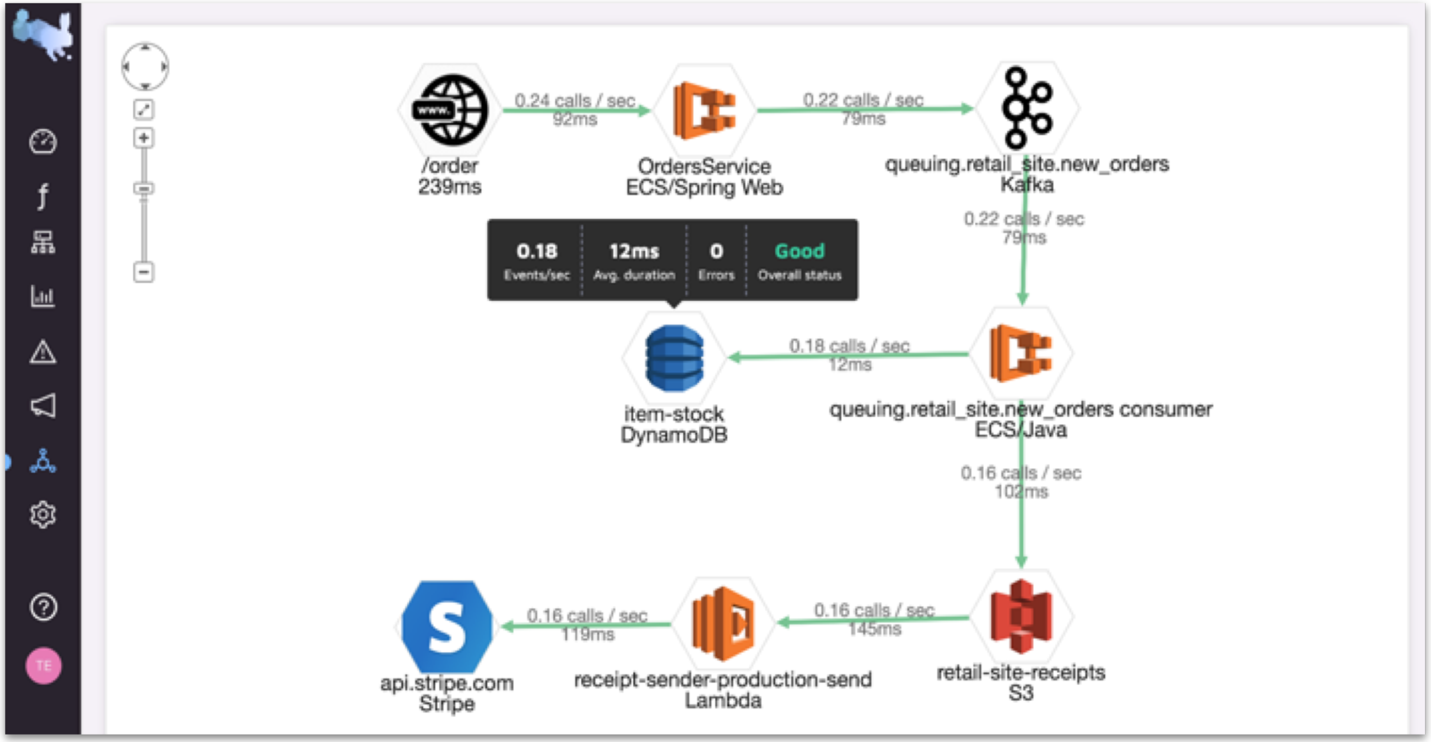
- trace -> logs
- trace <-> environment

epsagon

# Best Practices for Observability

# Best practices for observability

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



| 0.18 | 12ms | 0 | Good |
|---|---|---|---|
| Events/sec | Avg. duration | Errors | Overall status |

# The journey to observability

- Identify your business goals and architecture model
- Determine your approach: DIY or managed
- Trial observability solutions
- Make sure the new service integrates to your ecosystem
- Evaluate the benefit and influence decision-makers

# Summary

- Modern applications requires more than just monitoring
- Distributed tracing becomes a crucial component in such environments
- Stop implementing your own solutions unless needed

# Thank you!

epsagon.com
@ranrib