



Jaeger Project Intro

Pavol Loffay (Red Hat), Yuri Shkuro (Uber)

CloudNativeCon NA, Seattle, Dec-12-2018

Agenda

- What is tracing
- Demo
- Project status
- New Features
- Roadmap
- Q & A

About

- Pavol Loffay (<https://github.com/pavolloffay>)
 - Software engineer at Red Hat
 - Working on tracing & observability
- Yuri Shkuro (<https://github.com/yurishkuro>)
 - Software engineer at Uber
 - Working on tracing & observability



CLOUD NATIVE
COMPUTING FOUNDATION

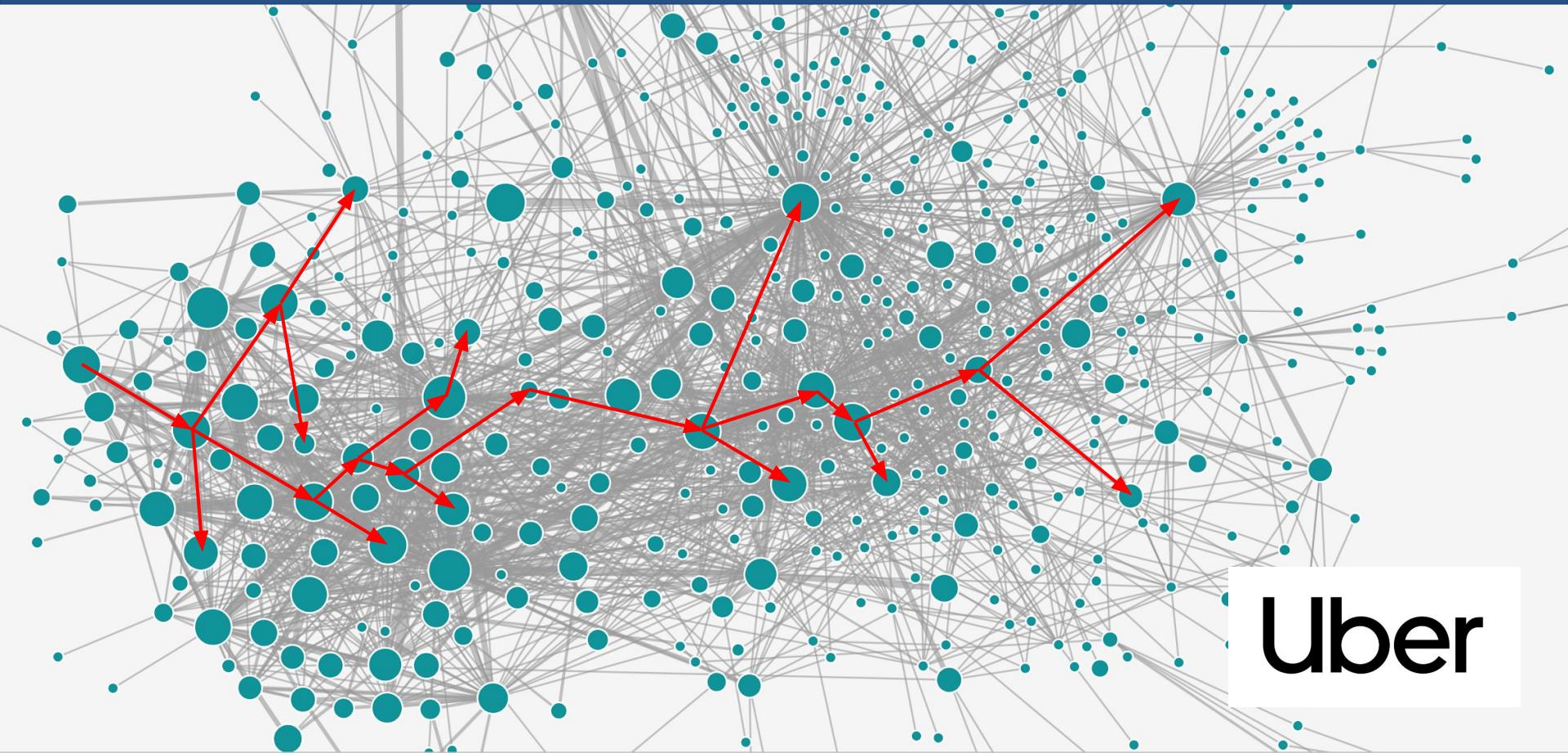
What is Tracing & Why?

Concepts and terminology

Modern Distributed Systems are COMPLEX

Loading Netflix or Facebook home page ⇒
dozens of microservices, 100s of nodes

BILLIONS of times a day!

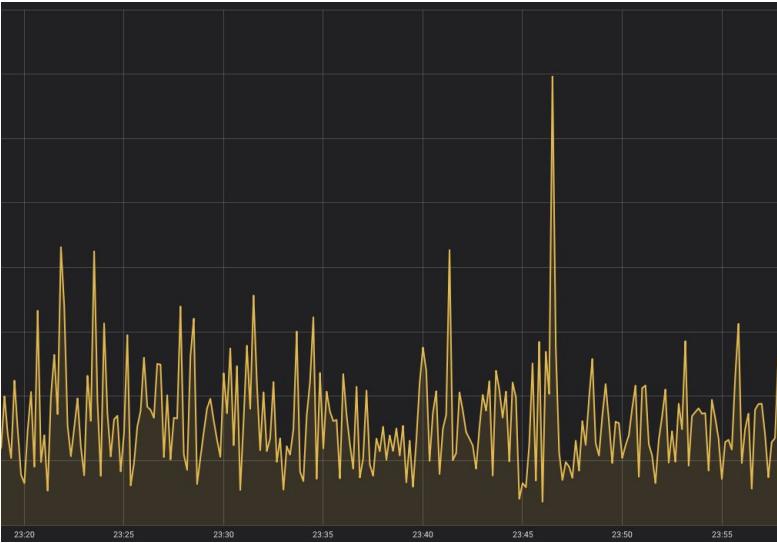


Uber

How can we tell what is going on?

Which service is to blame
when things go wrong or become slow?

Traditional monitoring tools don't help



Metrics show something is wrong, but do not explain why.

Logs are a mess: concurrent requests, multiple hosts, impossible to correlate.

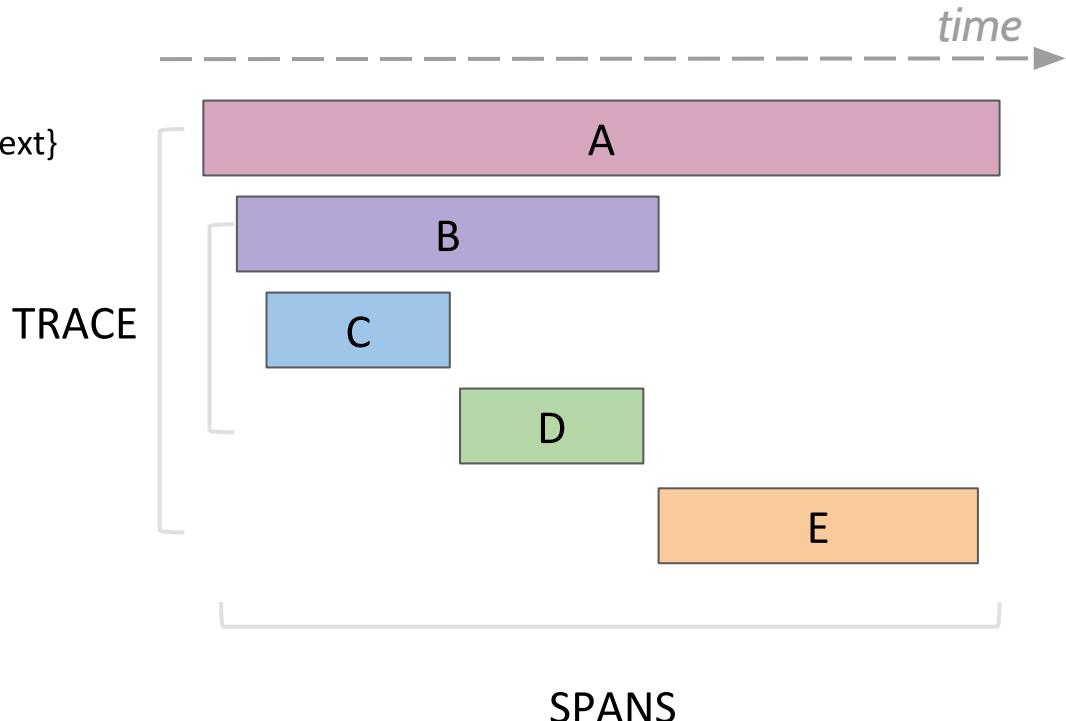
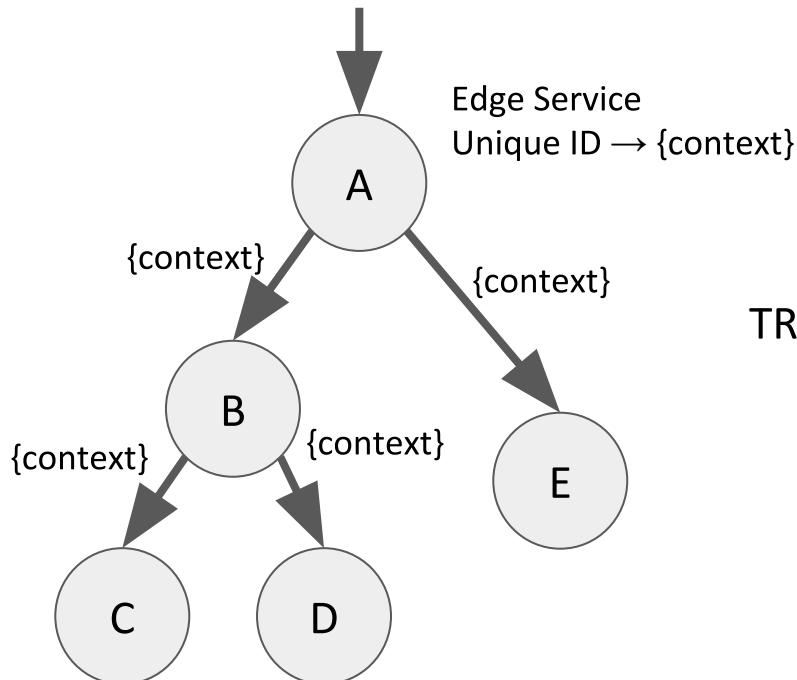
Monitoring tools must tell stories!

Do you like debugging
without a stack trace?

We need to monitor
distributed transactions
⇒ **distributed tracing!**



Context Propagation & Distributed Tracing



Jaeger, a Distributed Tracing Platform

trace collection
backend

visualization
frontend

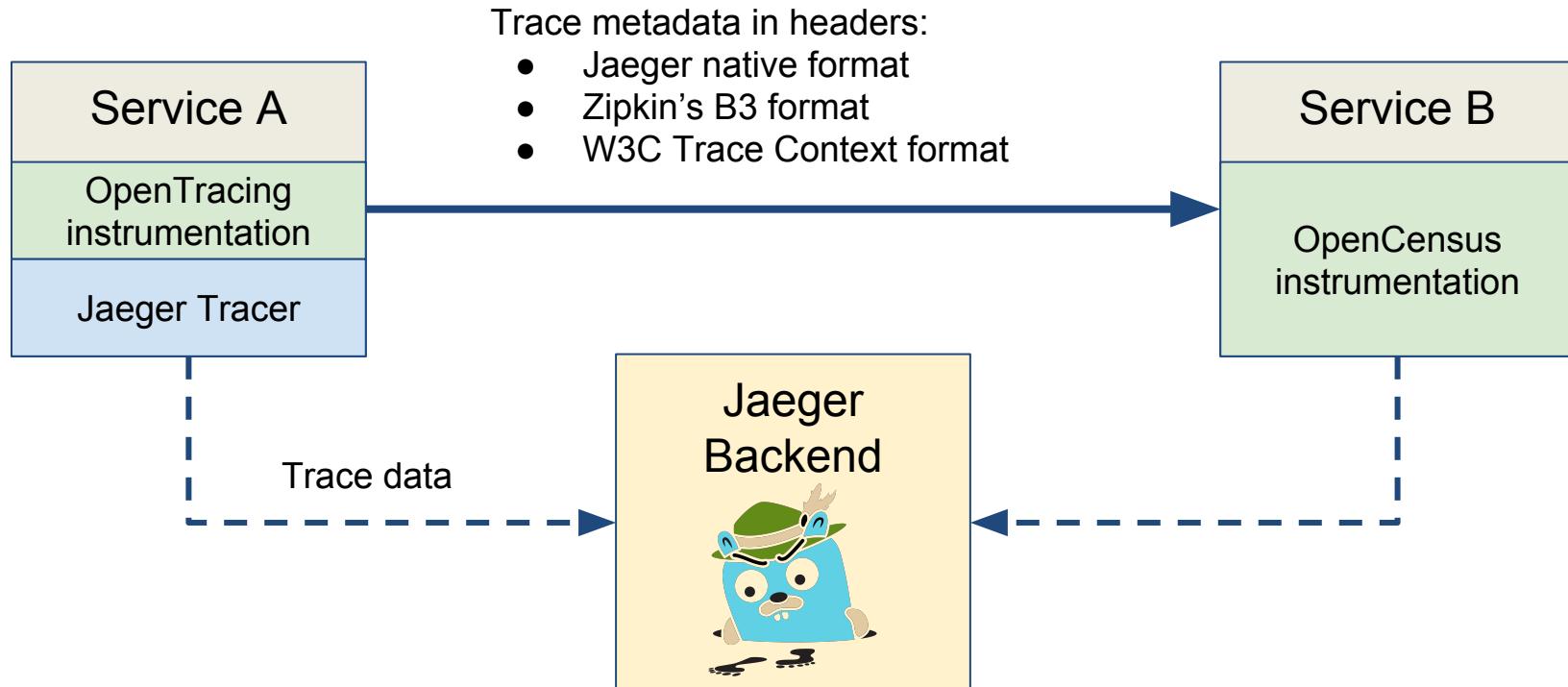
instrumentation
libraries

data mining
platform



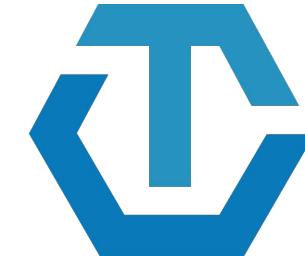
<https://jaegertracing.io>

Jaeger Integrations



OpenTracing

- **Instrumentation API**
 - Context propagation
 - Distributed tracing
 - Contextualized logging
 - Contextualized metrics
- Vendor neutral
- Cross language
- CNCF member project



OPENTRACING

<http://opentracing.io>



CLOUD NATIVE
COMPUTING FOUNDATION

Jaeger - /'yāgər/, noun: hunter

- Inspired by Google's Dapper and OpenZipkin
- Started at Uber in August 2015
- Open sourced in April 2017
- Joined CNCF in Sep 2017 (incubating)
- Applying for graduation

<https://github.com/cncf/toc/pull/171>



Technology Stack

- Go backend
- Pluggable storage
 - Cassandra, Elasticsearch, memory, ...
- React/Javascript frontend
- OpenTracing Instrumentation libraries
- Integration with Kafka, Apache Flink



Go



Java™
POWERED

python
powered



Project & Community

- 7 maintainers, from Uber and Red Hat
- GitHub stats
 - >6,600 stars, >880 forks
 - >580 contributors
 - >220 authors of commits and pull requests
 - >350 issue creators





Let's look at some traces

demo time: <http://bit.do/jaeger-hotrod>



Distributed Tracing Systems

distributed
transaction
monitoring

performance
and latency
optimization

root cause
analysis

service
dependency
analysis

distributed context propagation



Jaeger 1.8 - 1.9

New Features



New Features

- New website, distributions
- Graph visualizations, trace diffs
- Integrations with other projects
- Async ingestion
- Protobuf & gRPC
- Better Zipkin compatibility

New Website (easy to contribute)

The screenshot shows a web browser window with the URL `jaegertracing.io` in the address bar. The page features a large blue hexagonal background pattern. In the top left corner is a cartoon owl logo wearing a beret, next to the word "JAEGER". The top right contains navigation links: "Download", "Docs", "Blog", "Project", and social media icons for GitHub and Twitter. Below the header, the text "Jaeger: open source, end-to-end distributed tracing" is displayed, followed by the subtitle "Monitor and troubleshoot transactions in complex distributed systems". At the bottom of the main section are three buttons: "GET STARTED" (with a play icon), "DOWNLOAD" (with a cloud icon), and "OCTOBER 2018 NEWSLETTER" (with a megaphone icon).

Why Jaeger?

As on-the-ground microservice practitioners are quickly realizing, the majority of operational problems that arise when moving to a distributed architecture are ultimately grounded in two areas: **networking** and **observability**. It is simply an orders of magnitude larger problem to network and debug a set of intertwined distributed services versus a single monolithic application.

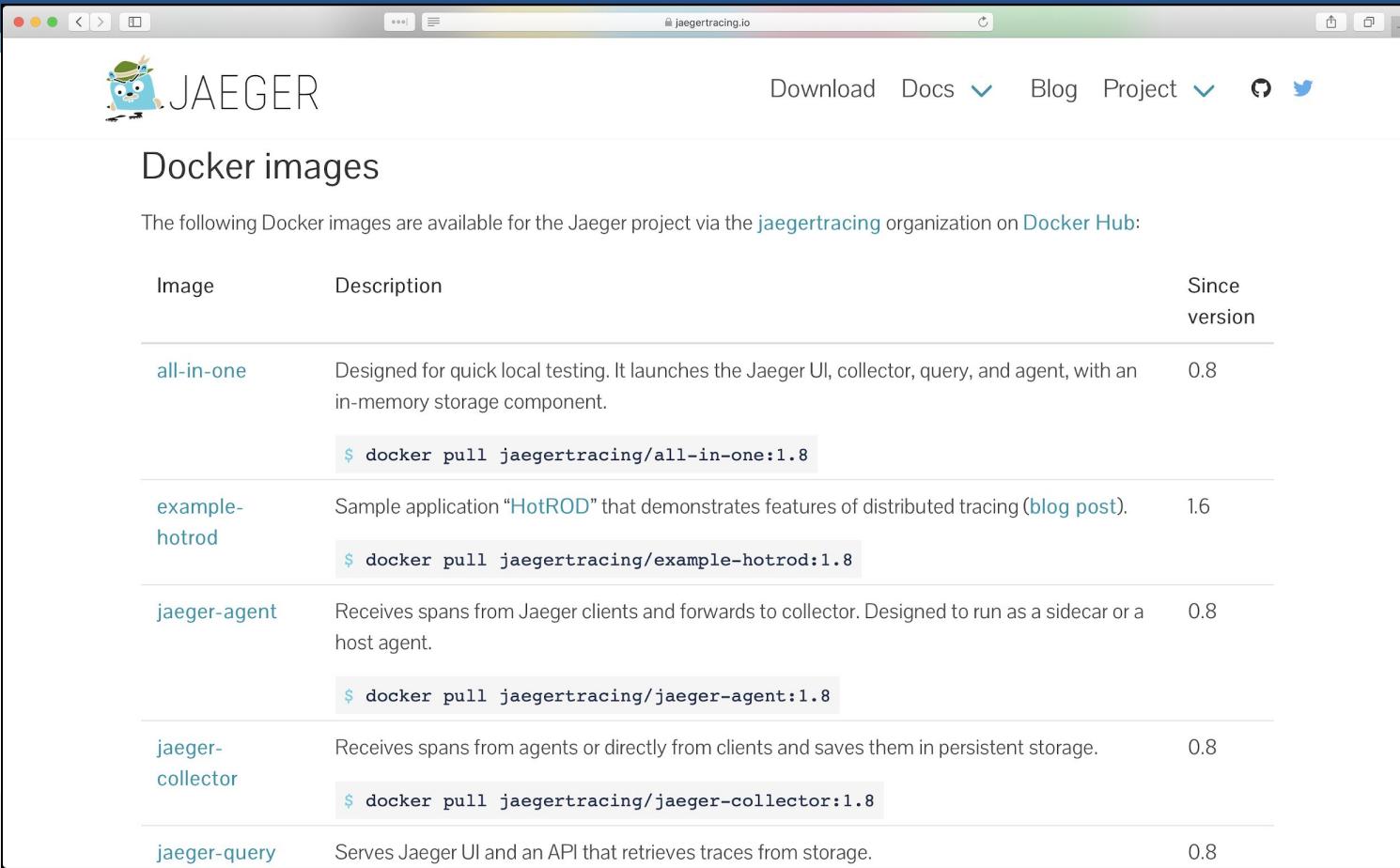
Example: Client Features matrix

The screenshot shows a web browser window for jaegertracing.io. The page title is "Client features". On the left, there's a sidebar with links: Introduction, Getting started, Features (selected), Client libraries (with a sub-link to Client features), Architecture, Deployment, Monitoring, and Sampling. The main content area has tabs for "Version 1.8" and "Latest" (selected). A note says: "The table below provides a feature matrix for the existing client libraries. Cells marked with ? indicate that it's not known if the given client supports the given feature and additional research & documentation update is required." Below this is a section titled "Data format and transport for reporting spans to Jaeger backend" with a table:

Feature	Go	Java	Node.js	Python	C++	C#
Report <code>jaeger.thrift</code> over UDP	✓	✓	✓	✓	✓	✓
Report <code>jaeger.thrift</code> over HTTP	✗	✓	✗	✗	?	✓
Report Zipkin Thrift over HTTP	✓	✗	✗	✗	✗	✗

At the bottom, there's a link: "Inter-process propagation wire format (headers)".

Distribution: Docker images

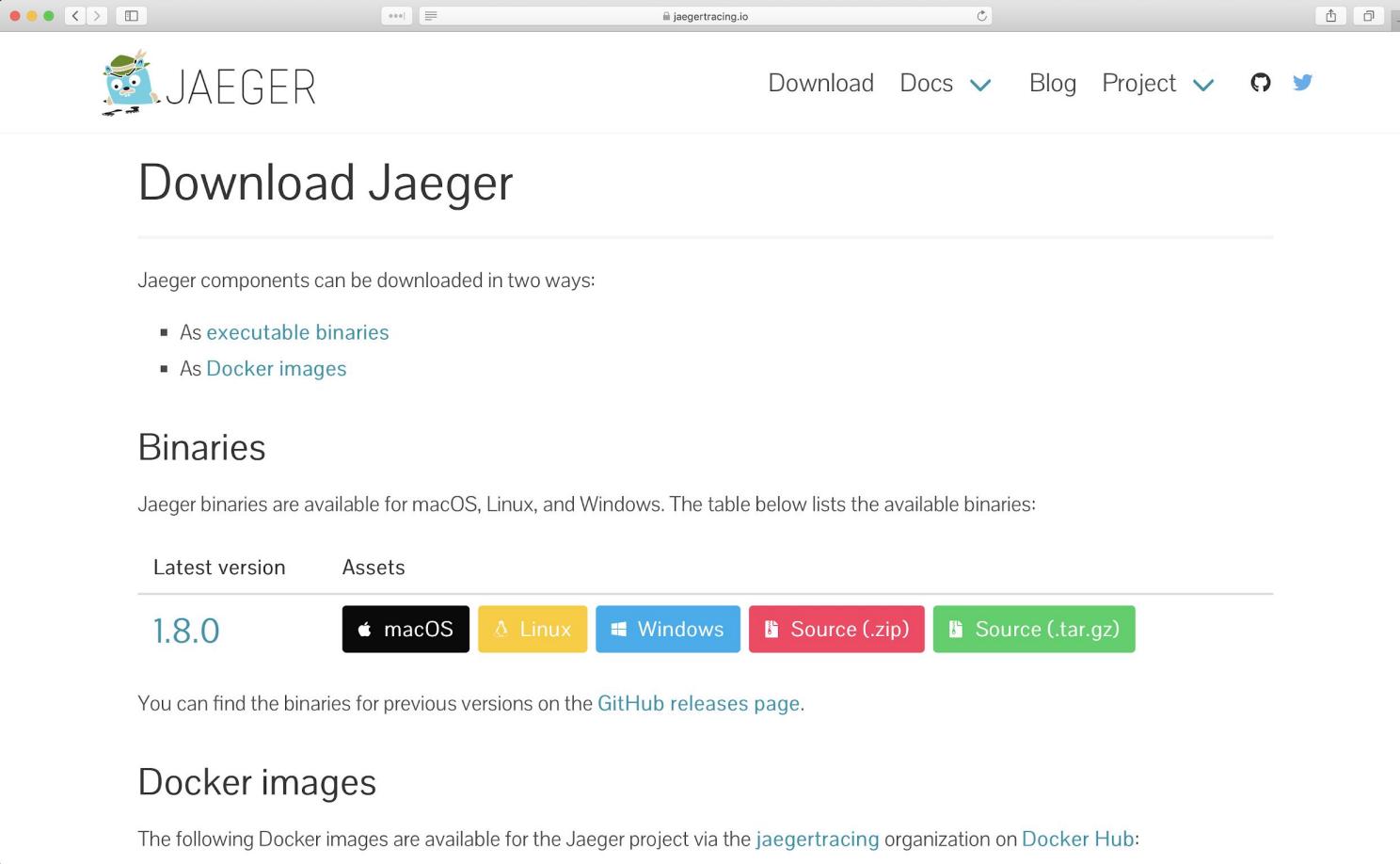


The screenshot shows a web browser window with the URL `jaegertracing.io` in the address bar. The page is titled "Docker images". It features a logo of a blue owl wearing a green hat and holding a sword. The navigation menu includes "Download", "Docs", "Blog", "Project", and social media links for GitHub and Twitter.

The main content area displays a table of Docker images available from the `jaegertracing` organization on Docker Hub:

Image	Description	Since version
<code>all-in-one</code>	Designed for quick local testing. It launches the Jaeger UI, collector, query, and agent, with an in-memory storage component. <code>\$ docker pull jaegertracing/all-in-one:1.8</code>	0.8
<code>example-hotrod</code>	Sample application "HotROD" that demonstrates features of distributed tracing (blog post). <code>\$ docker pull jaegertracing/example-hotrod:1.8</code>	1.6
<code>jaeger-agent</code>	Receives spans from Jaeger clients and forwards to collector. Designed to run as a sidecar or a host agent. <code>\$ docker pull jaegertracing/jaeger-agent:1.8</code>	0.8
<code>jaeger-collector</code>	Receives spans from agents or directly from clients and saves them in persistent storage. <code>\$ docker pull jaegertracing/jaeger-collector:1.8</code>	0.8
<code>jaeger-query</code>	Serves Jaeger UI and an API that retrieves traces from storage. <code>\$ docker pull jaegertracing/jaeger-query:1.8</code>	0.8

Binaries (Linux, MacOS, Windows)



The screenshot shows a web browser window displaying the Jaeger tracing website at jaegertracing.io. The page has a blue header with the text "Binaries (Linux, MacOS, Windows)". Below the header is a navigation bar with links for "Download", "Docs", "Blog", "Project", and social media icons for GitHub and Twitter. On the left, there's a logo of a blue owl wearing a green beret, labeled "JAEGER". The main content area has a heading "Download Jaeger" and a sub-section titled "Binaries". It states that components can be downloaded as executable binaries or Docker images. A table lists the latest version (1.8.0) and assets available for macOS, Linux, Windows, Source (.zip), and Source (.tar.gz). A note at the bottom says you can find previous versions on the GitHub releases page.

Download Jaeger

Binaries

Jaeger binaries are available for macOS, Linux, and Windows. The table below lists the available binaries:

Latest version	Assets
1.8.0	macOS Linux Windows Source (.zip) Source (.tar.gz)

You can find the binaries for previous versions on the [GitHub releases page](#).

Docker images

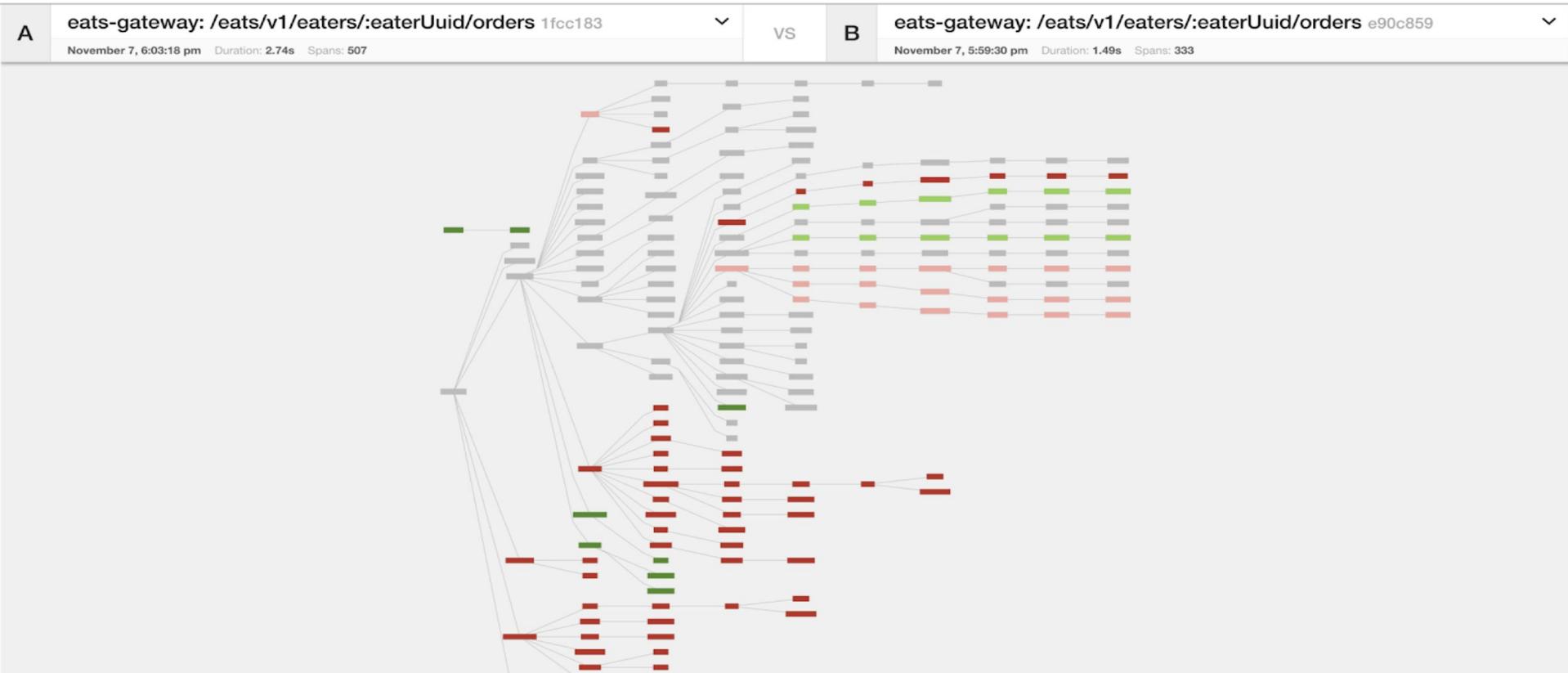
The following Docker images are available for the Jaeger project via the [jaegertracing](#) organization on Docker Hub:

Graph Visualizations

Gantt chart is not great for traces with 10s of thousands of spans

- Trace Diffs
 - Compare two traces
 - Compare one trace against a group of traces (coming soon)
- Trace Graph (coming soon)
 - Call graph visualization with mini-aggregations
 - Showing paths rather than individual RPCs

Comparing trace structures – Unified diff



Graph Visualizations

- Surface less information
- Condense the structural representation
- Emphasize the differences
- Distinct comparison modes simplify the comparisons

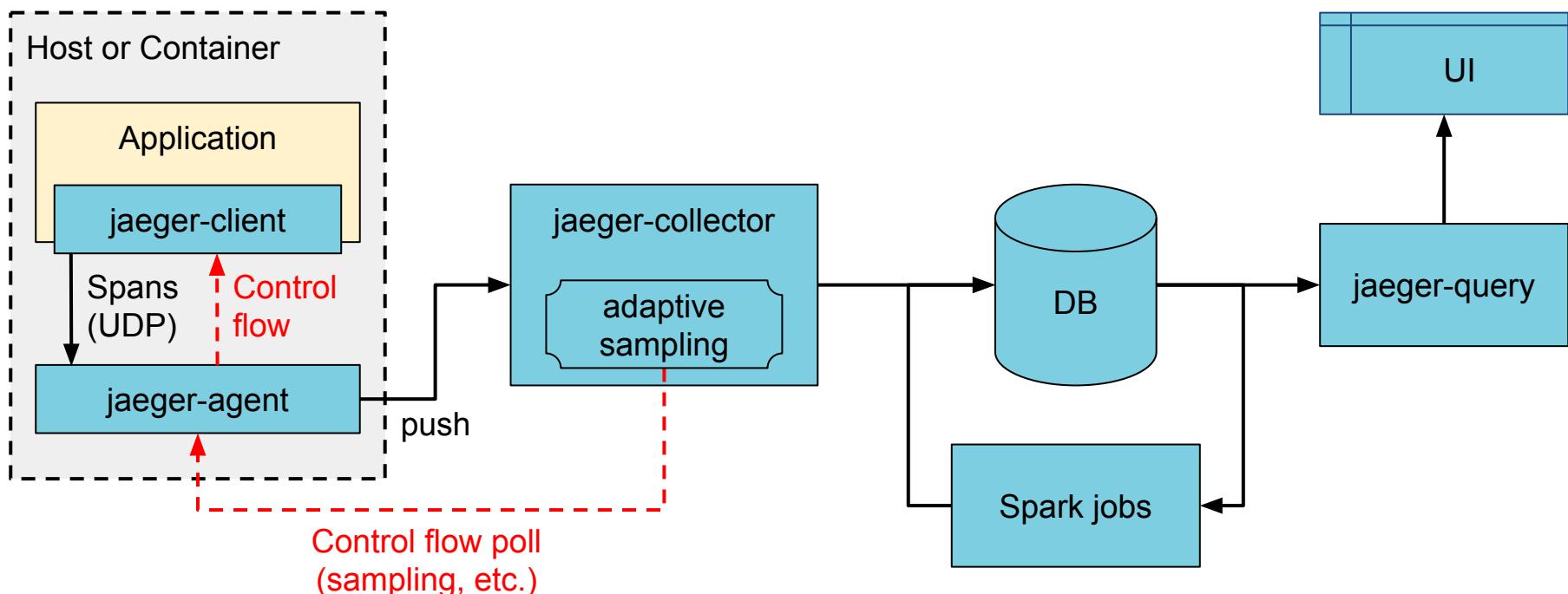
Integrations

- Jaeger Operator for Kubernetes
 - <https://github.com/jaegertracing/jaeger-operator>
- OpenCensus libraries and agent ship with exporters for Jaeger
 - <https://opencensus.io/guides/exporters/supported-exporters/java/jaeger/>
- Istio comes with Jaeger included
 - <https://istio.io/docs/tasks/telemetry/distributed-tracing/>
- Envoy works with Jaeger native C++ client
 - https://www.envoyproxy.io/docs/envoy/latest/start/sandboxes/jaeger_native_tracing
- Eclipse Trace Compass incubator supports importing Jaeger traces
 - <https://github.com/tuxology/tracevizlab/tree/master/labs/303-jaeger-opentracing-traces>

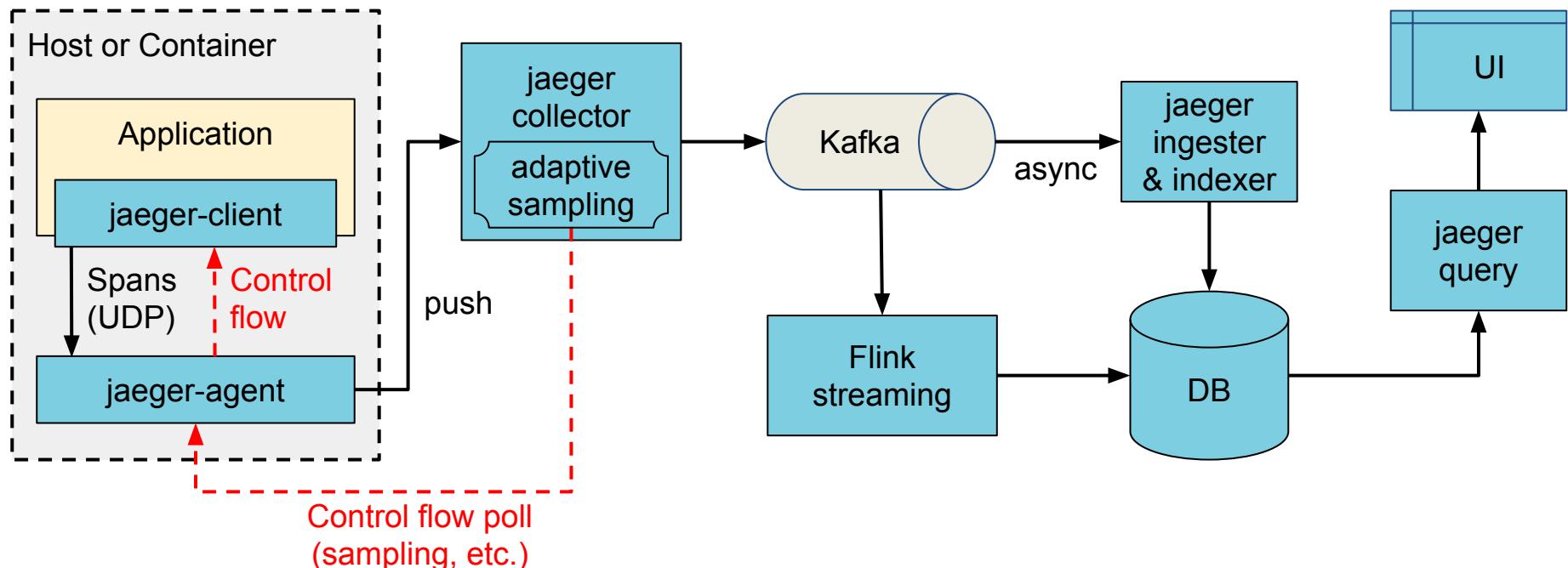
Asynchronous span ingestion

- Push model was struggling to keep up with traffic spikes
 - Because of sync storage writes
 - Collectors had to drop data randomly
- Kafka is much more elastic for writes
 - Just raw bytes, no schema, no indexing
 - A lot less overhead on the write path
- Data in Kafka allows for streaming data mining & aggregations
- Two new components: **jaeger-ingester** and **jaeger-indexer**

Architecture 2017: Push



Architecture now: Push+Async+Streaming



Protobuf & gRPC

- Internal data model generated from Protobuf IDL
- gRPC connection between `jaeger-agent` and `jaeger-collector`

Why

- gRPC plays better with modern routing than TChannel
- Path to official data model and collector/query APIs
- Protobuf-based JSON API
- Unblock development of storage plugins
- (Thrift still supported for backwards compatibility)

Zipkin Compatibility

- Clients
 - Zipkin B3-*** headers for context propagation
 - Interop between Jaeger-instrumented and Zipkin-instrumented apps
- Collector
 - Zipkin Thrift and JSON v2 span format
 - Use Zipkin instrumentation (e.g. Brave) to send traces to Jaeger
- Outstanding
 - Accept Zipkin spans from Kafka stream



CLOUD NATIVE
COMPUTING FOUNDATION

Roadmap

<http://bit.do/jaeger-roadmap>



Adaptive Sampling

Problem

- APIs have endpoints with different QPS
- Service owners do not know the full impact of sampling probability

Adaptive Sampling is per service + endpoint,
decided by Jaeger backend based on traffic

Adaptive Sampling Status

- Jaeger clients support per service/endpoint sampling strategies
- Can be statically configured in collector
- Pull requests for dynamic recalculations

Data Pipeline

- Based on Kafka and Apache Flink
- Support aggregations and data mining
- Examples:
 - Pairwise dependencies diagram
 - Path-based dependencies diagram
 - Latency histograms



Storage plugins

- Based on gRPC/Protobuf work
- PRs in progress for proof of concept
- Community support for different storage backends



Partial Spans (community driven)

- Add ability to store/retrieve partial spans
- Use case:
 - Certain workflows are hours long. Unfortunately spans are only emitted once after it's Finished(). “Root span” is missing until the complete workflow is finished.



CLOUD NATIVE
COMPUTING FOUNDATION

Learn More

Website: jaegertracing.io/

Blog: medium.com/jaegertracing

Getting in Touch

- GitHub: <https://github.com/jaegertracing>
- Chat: <https://gitter.im/jaegertracing/>
- Mailing List - jaeger-tracing@googlegroups.com
- Blog: <https://medium.com/jaegertracing>
- Twitter: <https://twitter.com/JaegerTracing>
- Bi-Weekly Community Meetings

Q&A

- Jaeger Deep Dive - Thursday, Dec 14, 10:50am



<https://jaegertracing.io>

Happy Tracing!

Q & A

Jaeger Deep Dive
Thursday, Dec 14, 10:50am