



**CLOUD NATIVE**  
**COMPUTING FOUNDATION**

# 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



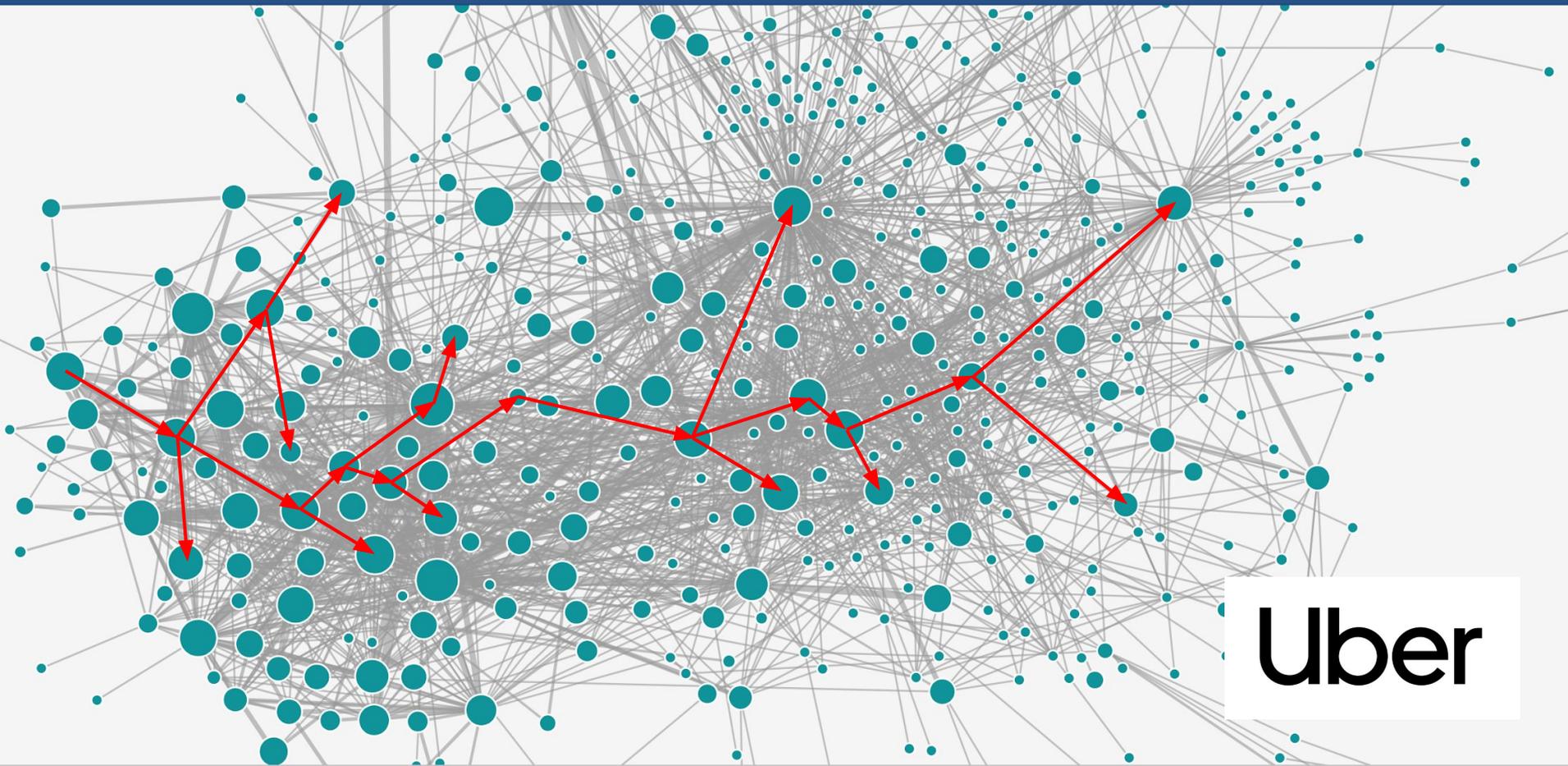
# What is Tracing & Why?

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?



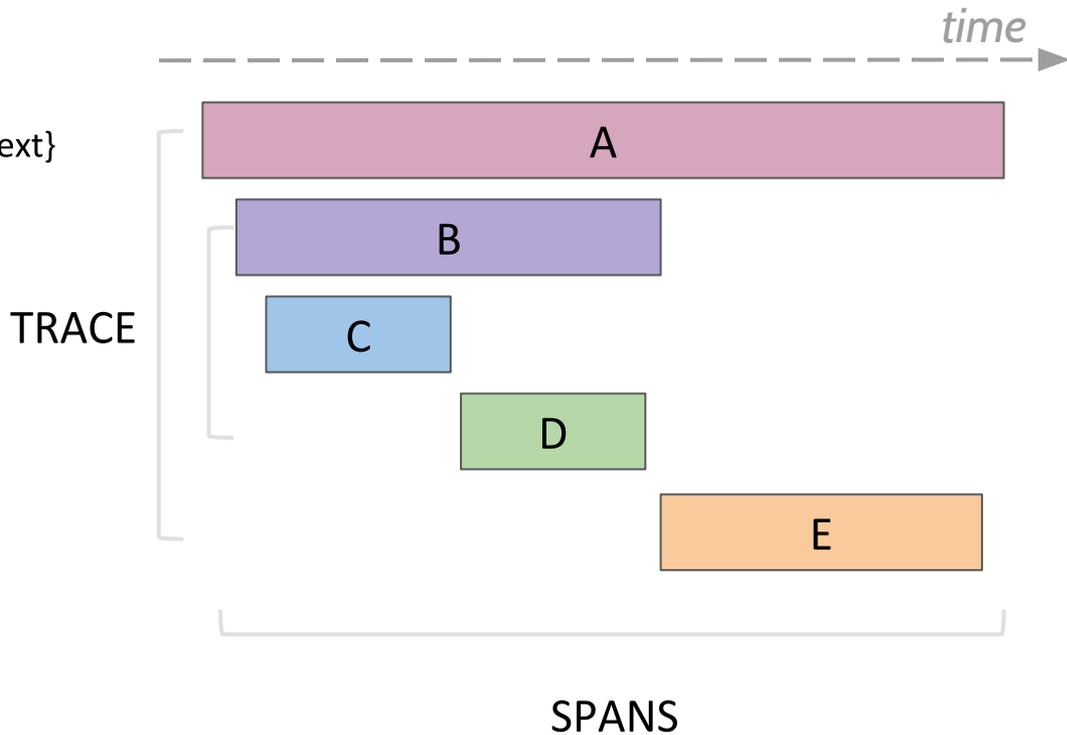
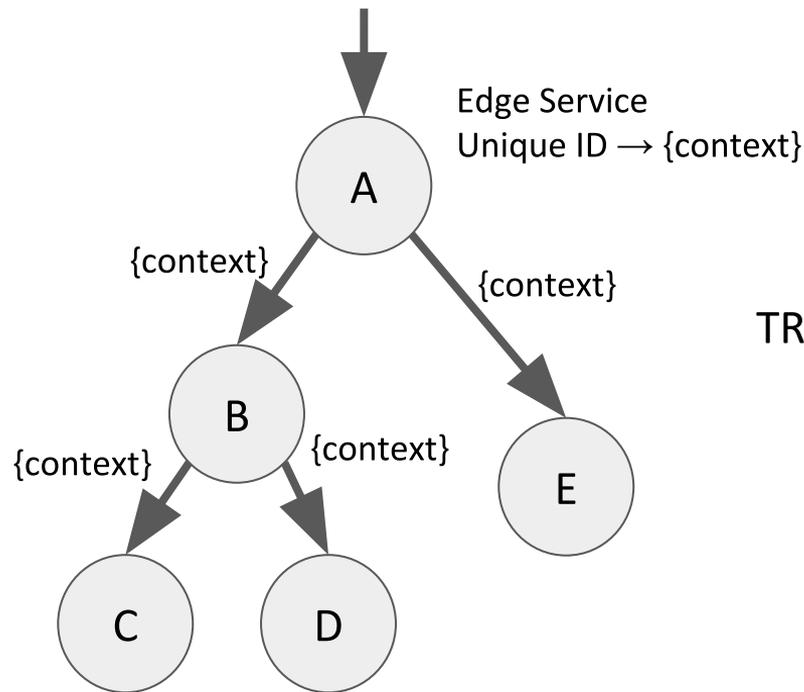
# Monitoring tools must tell stories!

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

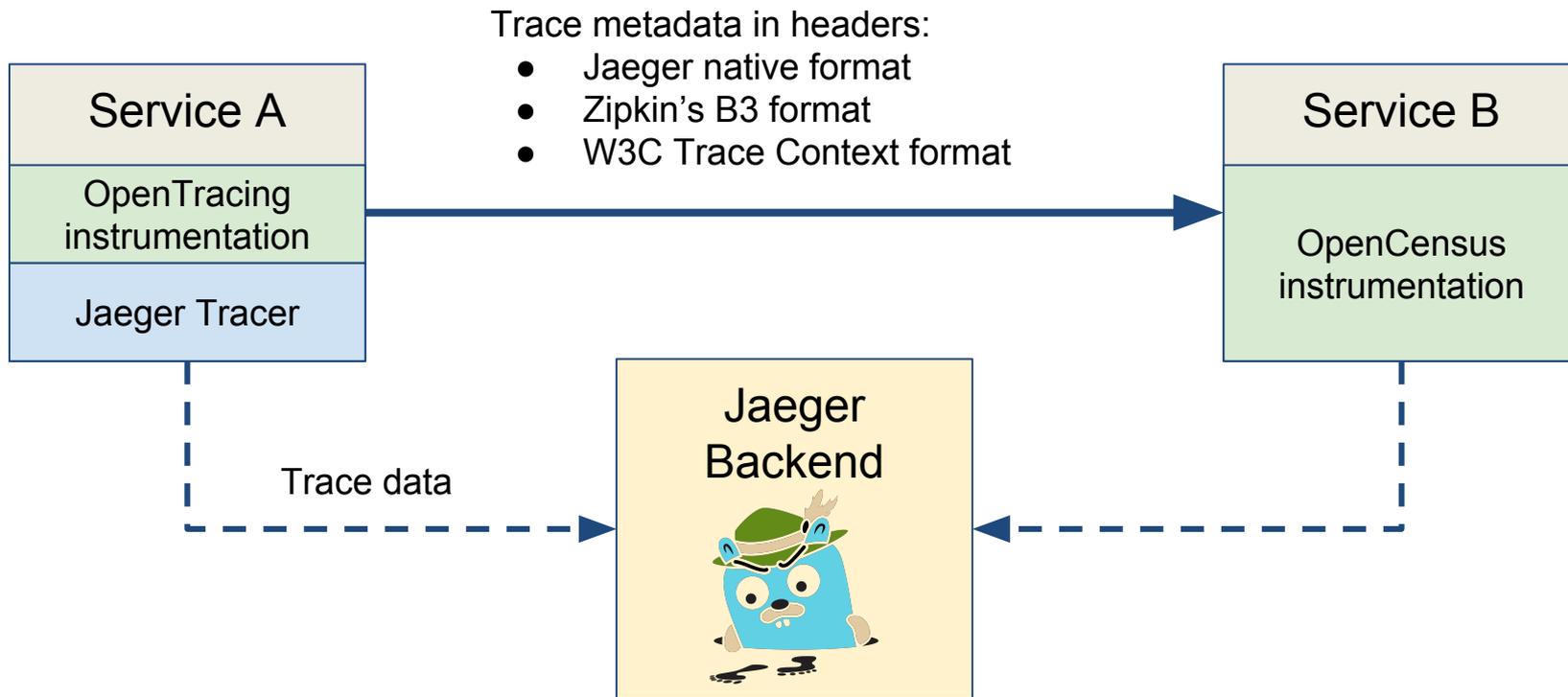
instrumentation  
libraries

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>



# 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



OPENTRACING



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

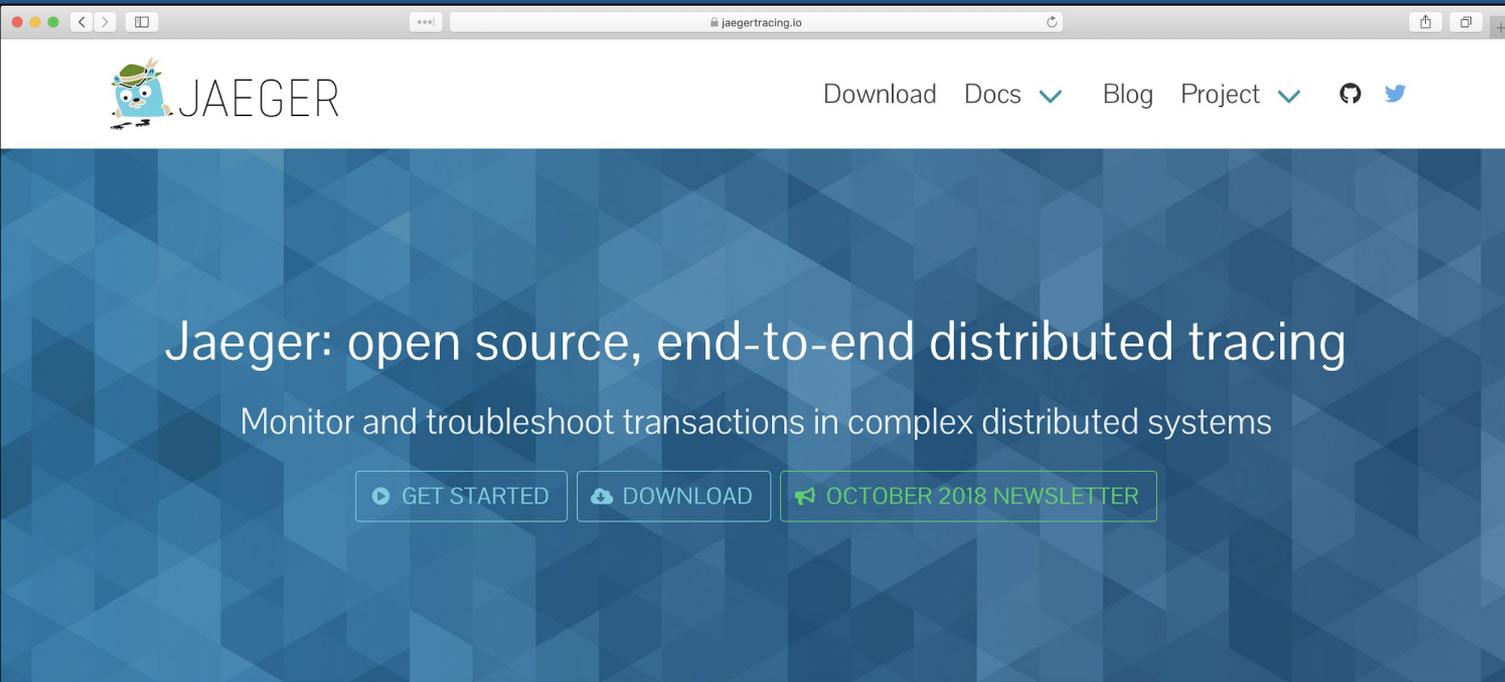
New Features



# New Features

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

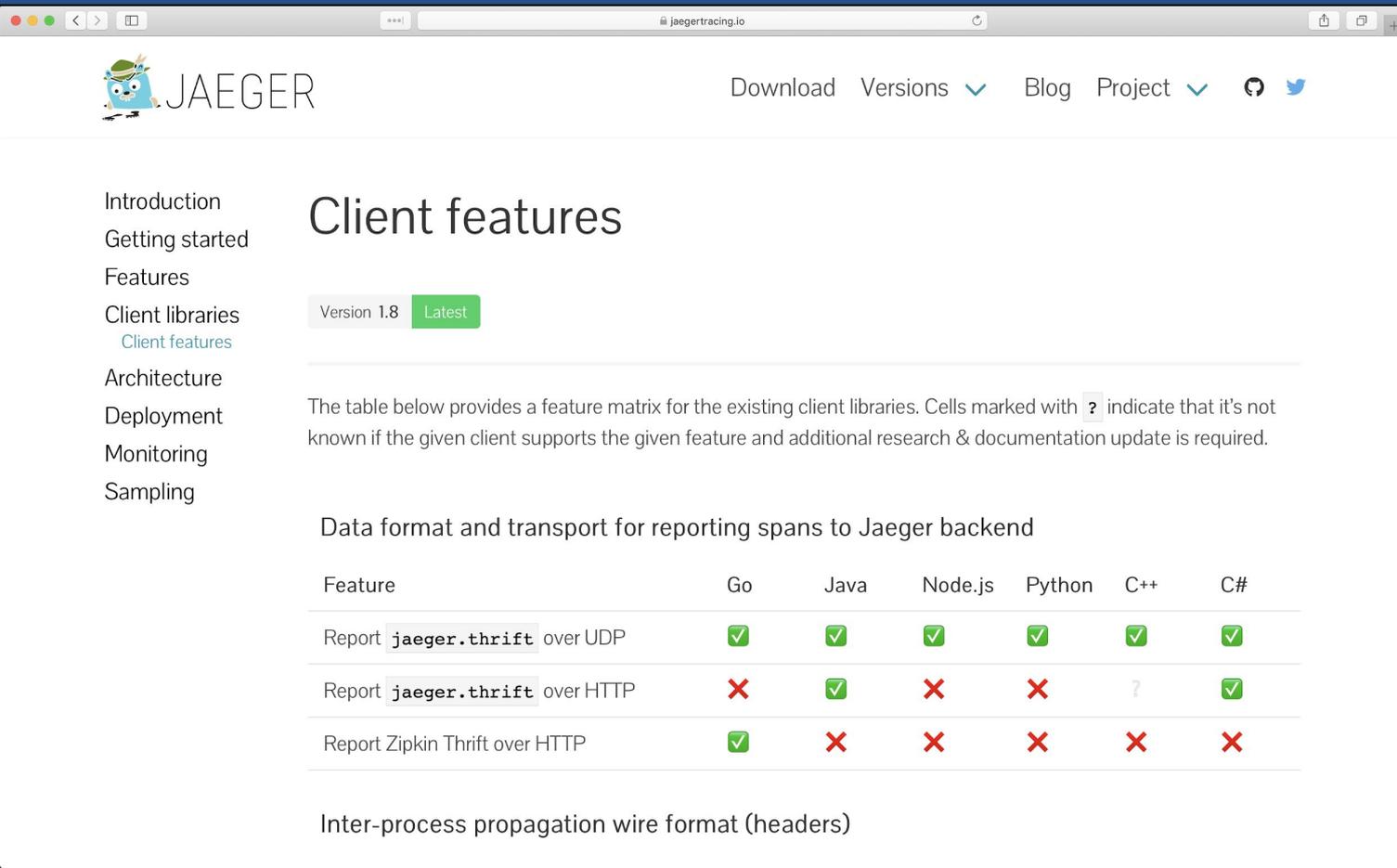
# New Website (easy to contribute)



## 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 the Jaeger website's 'Client features' page. The page header includes the Jaeger logo and navigation links for 'Download', 'Versions', 'Blog', and 'Project'. A sidebar on the left lists navigation options: 'Introduction', 'Getting started', 'Features', 'Client libraries', 'Architecture', 'Deployment', 'Monitoring', and 'Sampling'. The main content area is titled 'Client features' and shows 'Version 1.8' as the current version, with 'Latest' also indicated. A paragraph explains that the table below provides a feature matrix for existing client libraries, with '?' indicating unknown support. The table lists features for Go, Java, Node.js, Python, C++, and C#. The features are: 'Report jaeger.thrift over UDP', 'Report jaeger.thrift over HTTP', and 'Report Zipkin Thrift over HTTP'. The page also includes a section for 'Data format and transport for reporting spans to Jaeger backend' and a section for 'Inter-process propagation wire format (headers)'.

Version 1.8 Latest

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.

### Data format and transport for reporting spans to Jaeger backend

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	✓	✗	✗	✗	✗	✗

Inter-process propagation wire format (headers)

# Distribution: Docker images



Download Docs Blog Project

## Docker images

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

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

# Binaries (Linux, MacOS, Windows)

Download Jaeger

Jaeger components can be downloaded in two ways:

- As [executable binaries](#)
- As [Docker images](#)

## Binaries

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

Latest version	Assets
1.8.0	<a href="#">macOS</a> <a href="#">Linux</a> <a href="#">Windows</a> <a href="#">Source (.zip)</a> <a href="#">Source (.tar.gz)</a>

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

A	eats-gateway: /eats/v1/eaters/:eaterUid/orders 1fcc183 November 7, 6:03:18 pm Duration: 2.74s Spans: 507	VS	B eats-gateway: /eats/v1/eaters/:eaterUid/orders e90c859 November 7, 5:59:30 pm Duration: 1.49s Spans: 333
---	---	----	---



# Graph Visualizations

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

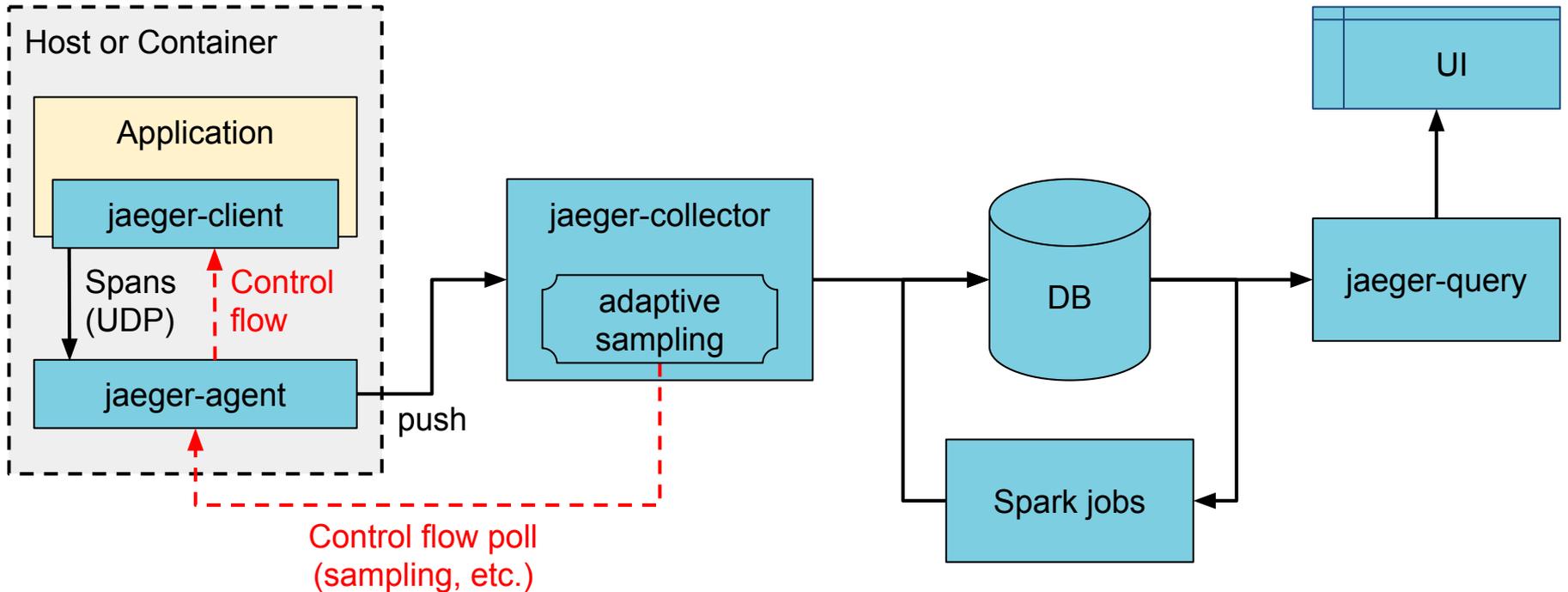
# Integrations

- 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](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>

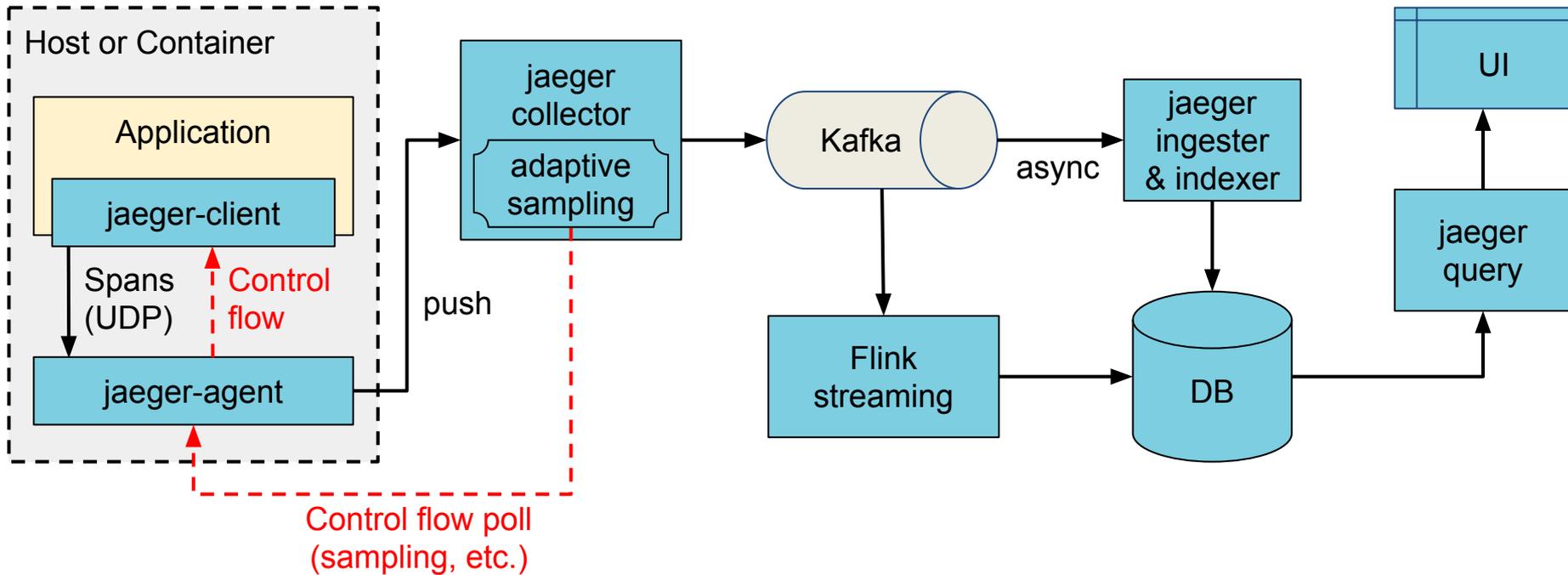
# Asynchronous span ingestion

- 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



# 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.



## Learn More

Website: [jaegertracing.io/](https://jaegertracing.io/)

Blog: [medium.com/jaegertracing](https://medium.com/jaegertracing)

# Getting in Touch

- GitHub: <https://github.com/jaegertracing>
- Chat: <https://gitter.im/jaegertracing/>
- [Mailing List](mailto:jaeger-tracing@googlegroups.com) - 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